UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

ALGORITMO DE SATISFACCIÓN PARA LÓGICAS MULTIMODALES CON INVERSAS

TESIS

QUE PARA OPTAR POR EL GRADO DE

MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:

DIEGO ROBERTO MEDINA MARTÍNEZ

TUTOR:

DR. ISMAEL EVERARDO BÁRCENAS PATIÑO

FACULTAD DE INGENIERÍA

CIUDAD UNIVERSITARIA, CD. MX., AGOSTO, 2024

PROTESTA UNIVERSITARIA DE INTEGRIDAD Y

HONESTIDAD ACADÉMICA Y PROFESIONAL

De conformidad con lo dispuesto en los artículos 87, fracción V, del Estatuto General, 68, primer párrafo, del Reglamento General de Estudios Universitarios y 26, fracción I, y 35 del Reglamento General de Exámenes, me comprometo en todo tiempo a honrar a ala institución y a cumplir con los principios establecidos en el Código de Ética de la Universidad Nacional Autónoma de México, especialmente con los de integridad y honestidad académica.

De acuerdo con lo anterior, manifiesto que el trabajo escrito titulado "Algoritmo de satisfacción para lógicas multimodales con inversas", que presenté para obtener el grado de Maestro en Ciencia e Ingeniería de la Computación, es original, de mi autoría y lo realicé con el rigor metodológico exigido por mi Programa de Posgrado, citando las fuentes, ideas, textos, imágenes, gráficos u otro tipo de obras empleadas para su desarrollo.

En consecuencia acepto que la falta de cumplimiento de las disposiciones reglamentarias y normativas de la Universidad, en particular las ya referidas en el Código de Ética, llevará a la nulidad e los actos de carácter académico administrativo del proceso de titulación/graduación

Atentamente

Diego Roberto Medina Martínez
No. de cuenta: 313115653

Agradecimientos

Expreso mi más profunda gratitud hacia mis queridos papás, por toda la dedicación y apoyo recibido, que ha influido en gran parte de mi vida, tanto personal como académica y profesional. Además, valoro la motivación e inspiración que me proporcionaron para poder superar todos los desafíos a los que me he enfrentado y, en consecuencia, alcanzar todas mis metas propuestas.

Quisiera agradecer a mis hermanos, por compartir conmigo su motivación y entusiasmo. Además, aprecio su interés en ayudarme a crecer como persona y su compañía, así como continuar trabajando hasta tarde cuando más lo necesitaba.

A Everardo por ser el mejor guía, por brindarme parte de su tiempo y proporcionarme los conocimientos necesarios para afrontar los desafíos que conllevó realizar este trabajo. Agradezco su compromiso porque todas y cada una de las reuniones que tuvimos fueron fructíferas y me encantó dialogar sobre un tema en el cual es experto, lo cual fue una gran fuente de motivación.

Agradezco al Consejo Nacional de Humanidades Ciencias y Tecnologías CONAHCYT la beca recibida (811727). Investigación realizada gracias al Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT) de la UNAM a través de los proyectos IA104122 y IA104724, de nombres “Algoritmos de razonamiento para lógicas no-clásicas” y “Verificación formal de sistemas inteligentes”. Agradezco a la DGAPA-UNAM la beca recibida.

Aclaración. Algunos resultados de este trabajo fueron publicados en [32], bajo el contexto del congreso CONISOFT 2023 (11th International Conference in Software Engineering Research and Innovation). Adicionalmente, algunos resultados fueron presentados en el taller LANMR 2023 (15th Latin American Workshop on New Methods of Reasoning).

Índice general

Índice de figuras

Índice de tablas

Introducción

En el campo de la teoría de la computación existen diversos problemas que han mantenido relevancia tanto académica como práctica a lo largo del tiempo, entre ellos se encuentra el problema de satisfacción. Este consiste en determinar si existe algún modelo que satisface a una fórmula [41]. La fórmula se expresa en un lenguaje formal el cual pertenece a un sistema lógico.

A manera de ilustración, un modelo se define en la lógica proposicional como la asignación booleana de valores a las variables, tal que si al ser evaluados sobre una fórmula, esta se mantiene verdadera, se concluye que la fórmula se satisface. Este caso se conoce como el problema de satisfacción booleana. Considere la siguiente expresión escrita en el lenguaje de la lógica proposicional: $p \rightarrow q$, un modelo que la satisface se presenta cuando ambas asignaciones de p y q son verdaderas.

El problema de satisfacción posee una diversidad de aplicaciones, principalmente porque los lenguajes formales suelen emplearse para expresar o modelar fragmentos de la realidad de manera precisa, lo que facilita razonar sobre esta. Por ejemplo, considere dos proposiciones: p como "hay conexión a internet" y q como "el cable de red está conectado", bajo el contexto de una computadora de escritorio que solo puede acceder a internet mediante una conexión por cable.

La fórmula $\neg p \wedge q$ se puede leer como "no hay conexión a internet y el cable de red está conectado". El único caso donde la fórmula se satisface se presenta cuando p es falsa y q es verdadera. Esto tiene sentido en el mundo real puesto que a pesar de que el cable de red está conectado, puede ser que no exista conexión a internet debido a problemas externos, como ilustración, problemas relacionados con el proveedor de internet.

Con lógicas más expresivas se pueden modelar problemas de mayor grado de complejidad. En particular la lógica modal proposicional posee dos nuevos operadores: $\Diamond$ para indicar el concepto de posibilidad y $\Box$ el concepto de necesidad. Para ilustrarlos, considere la fórmula $(\Diamond\neg p \wedge \Box q) \wedge (\Diamond p \wedge \Box q)$ la cual se puede leer como "Es posible que no hay conexión

a internet y es necesario que el cable de red está conectado, y es posible que hay conexión a internet y es necesario que el cable de red está conectado.".

En el ejemplo se observa el modelado de dos situaciones posibles, una donde es posible que haya conexión a internet y otra donde no lo es. En ambas es necesario que el cable de red está conectado. Ambas situaciones suceden en la realidad y se puede observar que con estos nuevos operadores es más natural la creación e interpretación de las fórmulas.

Por otro lado, en un lenguaje que posee múltiples modalidades e inversas, se pueden modelar aún más problemas. A modo de ejemplo, considere un contexto en el que se tiene un sistema de comunicación cifrado donde las modalidades m corresponden a los distintos canales de comunicación disponibles, por lo que se puede representar: "enviado mediante el canal m" y la modalidad inversa como "recibido mediante el canal m". Entonces, cada modalidad es un medio distinto de comunicación, la modalidad sin inversa expresa "recibido" y la modalidad con inversa "enviado."

Adicionalmente, considere la proposición p como un mensaje que afirma "la contraseña es 123" y la proposición q como "el acceso al sistema se concede". Entonces, la fórmula $\boxed{1}p \leftrightarrow \diamondsuit_{\overline{1}} q$ se lee como "es necesario enviar mediante el canal 1 que la contraseña es 123, si y solo si, es posible recibir mediante el canal 1 que el acceso al sistema se concede".

En el ejemplo se emplea la modalidad 1 y $\overline{1}$ para indicar que se envían y reciben mensajes mediante el canal 1, respectivamente. Se considera un escenario en el que se envía un mensaje que contiene determinadas credenciales, a cambio es posible recibir un mensaje indicando que el acceso se concede, debido a que no se desea exponer el intento fallido cuando la contraseña no es correcta.

Con el objetivo de ilustrar algunas aplicaciones más concretas del problema de satisfacción, en el caso de las lógicas modales, existe un trabajo que propone validar la consistencia de los sistemas conscientes de la atención [28]. Su objetivo consiste en verificar un conjunto de reglas pertenecientes a un sistema de comunicación bidireccional, empleando un algoritmo de satisfacción para cálculo μ [25].

Por otra parte, existen trabajos relacionados a la verificación de la consistencia de bases de datos [10], [11]. En estos trabajos se utiliza un algoritmo de satisfacción para lógica proposicional, con el propósito de identificar inconsistencias bajo las restricciones existentes en un esquema de base de datos relacional, por ejemplo, violación de las llaves primarias, llaves foráneas, valores únicos, entre otros.

Para identificar las inconsistencias se emplean consultas típicas a una base de datos como entrada, las cuales son modeladas en lógica de primer orden. Además, se genera una fórmula escrita en lógica proposicional que modela la base de datos, se corrobora la satisfacción de esta última con el propósito de encontrar posibles inconsistencias en la consulta, de manera que se analizan los valores de las variables proposicionales que satisfacen la

fórmula booleana.

En este trabajo se analizan las técnicas y los algoritmos de satisfacción de las lógicas modales encontradas en el estado del arte. Posteriormente, se presenta la lógica multimodal K con inversas y la prueba de la propiedad de modelo de árbol para esta lógica. Esta propiedad se emplea en el algoritmo con el objetivo de operar con estructuras libres de ciclos y que garantizan una inserción de nodos ordenada. A su vez, facilita trabajar bajo profundidad o anchura.

Subsecuentemente, comienza la definición del algoritmo y la especificación de las funciones principales que lo componen, así como algunos ejemplos. Una vez establecido el algoritmo se realiza la prueba de la corrección, que involucra demostrar la coherencia y la completitud.

Después de la definición formal del algoritmo y que se ha probado que es correcto, se realizó la implementación. Esta etapa implicó analizar el lenguaje de programación a emplear, determinar los tipos de estructuras de datos necesarias y considerar las posibles optimizaciones a nivel de código, por ejemplo, el uso de memoria dinámica como los apuntadores para el manejo de las fórmulas en los nodos de la estructura.

Durante el desarrollo del sistema se consideraron buenas prácticas, así como un marco de trabajo iterativo, focalizado en la revisión continua del sistema. Finalmente, se llevaron a cabo los experimentos, dado un conjunto de fórmulas previamente definido, con el objetivo de realizar una medición cuantitativa del rendimiento del algoritmo frente a determinados casos.

Capítulo 1

Antecedentes

La presente sección se enfoca en analizar algunos de los algoritmos de satisfacción para lógicas modales y sus optimizaciones, así como señalar las áreas de oportunidad que se abordan en este trabajo. En el estado del arte, existe una diversidad de técnicas para afrontar el problema de satisfacción. En [33] los algoritmos se clasifican de acuerdo al enfoque con el que determinan la solución.

La primer estrategia se denomina construcción *Tableau*. Esta consiste en probar exhaustivamente una serie de reglas de manera recursiva. Este proceso comienza con la fórmula de entrada. A dicha fórmula se le aplican las reglas, lo que provoca que se creen nuevas ramificaciones, como si de un árbol se tratase. Este proceso continúa en las nuevas ramificaciones hasta que las fórmulas no se puedan descomponer más.

Finalmente, se explora el árbol y se buscan contradicciones. Si todas las ramificaciones conducen a contradicciones, se concluye que la fórmula no se satisface. En caso contrario, si existe al menos una ramificación que no contiene contradicciones, se concluye que la fórmula se satisface.

Por otro lado, existe otro método por traducción. Este consiste en reducir el problema de satisfacción de alguna lógica modal al problema de satisfacción booleana (SAT) o lógica de primer orden, por mencionar algunas lógicas objetivo. En contraste, existen otros que emplean una perspectiva basada en algoritmos SAT, como el algoritmo Davis-Putnam-Logemann-Loveland [9] (DPLL), el cual trata las subfórmulas modales como variables proposicionales. Algunos otros emplean técnicas específicas, por ejemplo, el enfoque por resolución, o algún otro basado en alguna propiedad de la lógica modal.

Adicionalmente, en esta investigación se identificaron algunos trabajos que utilizan de base un algoritmo SAT para resolver el problema en la lógica modal, que a diferencia del método de traducción tradicional, no se convierte directamente la fórmula, sino que el problema se divide en partes más pequeñas. Así, en el proceso de construcción de los posibles modelos, la traducción se aplica mediante el uso del algoritmo SAT, como paso

intermedio, considerando fragmentos del modelo en construcción. A continuación, se describen algunos de los algoritmos de satisfacción para lógicas modales con detalle.

1.1. Algoritmos de satisfacción para lógicas modales

En [7] se presenta un algoritmo de satisfacción para lógica modal $S5$ denominado S52SAT. Este emplea un enfoque de traducción a SAT. Una de las optimizaciones que consideran, consiste en reducir el tamaño de la fórmula codificada, de manera que el número de variables sea el menor posible. Para lograrlo establecen un límite superior con respecto al tamaño máximo de mundos para todos los posibles modelos, dada la fórmula de entrada. El límite está dado por una propiedad presentada en dicho trabajo como el grado de las modalidades diamante.

El grado de las modalidades diamante, a grandes rasgos, se define como una función recursiva que recibe de entrada una fórmula en forma normal negativa y retorna un valor numérico. Internamente posee un contador que comienza en cero y se incrementa en uno cada vez que se identifica una modalidad diamante. Para el caso de la conjunción se suma el grado de las modalidades diamante de ambos operandos. Por otro lado, en la disyunción se obtiene el valor máximo del grado de las modalidades diamante considerando los dos operandos.

Adicionalmente, utilizan un sistema de almacenamiento en caché para disminuir el consumo de memoria. El algoritmo fue implementado en OCaml y con esta implementación realizan una comparación con otros algoritmos de lógica modal $S5$, considerando el consumo de memoria y tiempo de ejecución. El objetivo de la experimentación consistió en cuantificar el rendimiento del algoritmo en conjunto con las técnicas de optimización implementadas.

Por otro lado, en [2] se presenta un algoritmo para lógica de conocimiento común. En este trabajo se emplea una técnica de construcción Tableau. Se destaca el uso de un único recorrido comparado con algoritmos que hacen dos iteraciones sobre el modelo que construyen, lo cual optimiza el tiempo promedio de ejecución. Realizaron un prototipo del algoritmo en un marco de trabajo llamado Tableau Work Bench [1], el cual permite construir rápidamente una implementación inicial de algoritmos de satisfacción con este enfoque de construcción.

Mientras tanto, en [30] se describe un algoritmo de satisfacción para lógica epistémica. El enfoque nuevamente es de construcción *Tableau*. Realizaron una implementación del algoritmo en el lenguaje de programación F#. Emplean memoria compartida con el objetivo de disminuir su consumo. Se menciona que este algoritmo funciona de manera aceptable con modelos relativamente pequeños.

Por otra parte, se ha desarrollado un algoritmo llamado K_m2SAT [38], [39] para la lógica multimodal K. Su técnica principal es la traducción a SAT y proponen una codificación de las fórmulas de entrada que disminuye el tamaño de la fórmula booleana generada. Las optimizaciones se concentran tanto en el preprocesamiento de la fórmula como en los pasos intermedios del algoritmo. Adicionalmente, efectuaron una implementación en C++, probaron el algoritmo en conjuntos de fórmulas conocidos y compararon los resultados con algunas herramientas del estado del arte.

Asimismo, existe un algoritmo de satisfacción para lógica multimodal K, llamado $KSAT$ [13]. En el trabajo se señala que los algoritmos que hacían uso de construcción *Tableau* en ese momento, usualmente eran menos eficientes con fórmulas grandes, comparado con su algoritmo que emplea un enfoque basado en un algoritmo de SAT llamado Davis-Putnam-Longemann-Loveland [9] (DPLL).

Adicionalmente, en $KSAT$ se menciona que los enfoques *Tableau* contienen información redundante en el modelo que construyen y que incrementa a medida que la profundidad de la fórmula crece, en contraste con DPLL que no genera ese tipo de información. Su implementación se realizó en Lisp y fue uno de los primeros algoritmos en tener un enfoque basado en algoritmos SAT, en el que los operadores modales se consideran variables proposicionales en cada nivel modal y se verifica que el modelo sea consistente con respecto a las fórmulas modales.

En la literatura se encuentra otro algoritmo de satisfacción, en este caso para lógica modal K, conocido como *SAT [12]. Este enfoque es orientado a SAT, específicamente se basa en las técnicas del algoritmo SATO [43]. Emplea una técnica para mejorar la eficiencia del algoritmo basada en caché, que a cambio de consumir más memoria llegan a la conclusión que obtienen ganancia en tiempo de ejecución acorde con sus experimentos.

Para razonar sobre lógica proposicional dinámica, lógica multimodal K, $K4$, KT y $S4$, se encuentra un algoritmo llamado DLP [18], [36]. Está basado en las técnicas de implementación de FaCT [17]. En esencia emplea una técnica de construcción *Tableau* y optimizaciones adicionales basadas en técnicas heurísticas, *backtracking*[1], caché y normalización de las fórmulas para evitar redundancia. Su implementación se realizó en el lenguaje ML. Comparado con sistemas de lógicas descriptivas y FaCT, este algoritmo fue en algunos casos mucho mejor en su momento.

Posteriormente, existe un algoritmo llamado $\mathcal{KBDD}$ [34], [35] que resuelve el problema de satisfacción para la lógica modal K mediante una perspectiva basada en autómatas finitos. Emplean diversas estrategias de optimización, entre ellas el uso de diagramas de decisión binarios (BDDs) y preprocesamiento de la fórmula, en esta última el objetivo

[1]Es una técnica que explora todas las posibles soluciones de un problema, de manera que en cada paso se determina si el camino que se está explorando tiene potencial para terminar en una solución correcta, en caso de no ser así se retrocede y se prueba otro camino.

consiste en disminuir el tamaño de la fórmula de entrada al eliminar redundancias, así como minimizar el número de operadores modales.

Adicionalmente, trabajan con dos enfoques: basados en tipos y partículas, al transformar la fórmula a su forma normal de la caja y forma normal negativa respectivamente, para luego codificarla en un BDD. La forma modal de la caja señala que debe aplicarse el operador de negación a variables proposicionales o a operadores modales caja. Además, hacen uso de la propiedad de modelo de árbol, lo que les permite crear y manipular autómatas con forma de árbol. Además posibilita una construcción de los modelos comenzando desde arriba hacía abajo y viceversa. Finalmente, se realizaron experimentos en los conjuntos TANCS 98 [4] y TANCS 2000 [31], así como una comparación con otros algoritmos: *SAT [12], DLP [36] y MSPASS [20].

Por otro lado, en la literatura está presente un algoritmo de satisfacción para cálculo μ que posee operadores de conteo y emplea modelos con forma de árbol [25]. El cálculo μ, en este caso, es una extensión de la lógica modal proposicional que añade operadores de punto fijo: μx conocido como punto fijo menor y νx como punto fijo mayor. Estos operadores permiten describir situaciones recursivas o infinitas y detallar cuándo una fórmula es verdadera o falsa en el contexto de los mundos accesibles.

En contraste con otras lógicas como la lógica de primer orden y de segundo orden, se presentan en este trabajo las ventajas de tener un algoritmo específico para el cálculo μ. De acuerdo con la información consultada hasta el momento, es el único algoritmo que existe para este tipo de lógica modal y su primera implementación en Java, por esta razón, no se realiza una comparación de este algoritmo con otros.

En el caso de la lógica modal graduada K_R, se identificó un algoritmo de satisfacción en [40] que puede manejar modalidades inversas. Este emplea un enfoque de construcción *Tableau* y su técnica de optimización principal consiste en determinar qué información será útil para los sucesores al momento de la generación del modelo y generar el sistema de restricciones en un enfoque de búsqueda en profundidad. Este trabajo únicamente describe el algoritmo, pero no realiza la implementación. Conforme a la información recabada, actualmente no existe alguna implementación de un algoritmo de satisfacción para la lógica modal graduada.

Se ha identificado un algoritmo de satisfacción para la lógica linear temporal (LTL) [24]. Este trabajo posee un enfoque basado en SAT, debido a que en esta lógica existen restricciones temporales que se transforman en restricciones booleanas, lo que permite producir un sistema de transición que ayuda a verificar la satisfacción en esta lógica. Se realiza su implementación y se realiza una comparación con un par de algoritmos existentes, entre ellos uno llamado TRP++ v.2.0 [19].

Está presente un algoritmo de satisfacción para lógica multimodal K [27] basado en la propiedad de modelo de árbol [14], [42]. Esta indica que existe una estructura de Kripke

Tabla 1.1: Algoritmos de satisfacción relacionados.

Algoritmo	Lógica	Método	Modalidades inversas	Implementación
[7]	Lógica modal $S5$	Traducción a SAT	No	OCaml
[38], [39]	Lógica multimodal K	Traducción a SAT	No	C++
[13]	Lógica multimodal K	Enfoque basado en SAT (DPLL)	No	Lisp
[12]	Lógica modal K	Enfoque basado en SAT (SATO)	No	C
[18], [36]	Lógica multimodal K	Construcción tableau	No	Standard ML
[34], [35]	Lógica modal K	Enfoque basado en autómatas finitos y la propiedad de modelo de árbol	No	C++
[17]	Lógica descriptiva	Construcción tableau	Sí	Lisp
[40]	Lógica modal graduada K_R	Construcción tableau	Sí	Ninguna
[27]	Lógica multimodal K	Propiedad de modelo de árbol	No	Java
Este trabajo	**Lógica multimodal K**	**Propiedad de modelo de árbol**	**Sí**	**Go**

y un mundo de esa estructura, desde el cual una fórmula de lógica modal se satisface si y solo si existe un modelo de tipo árbol que en su nodo raíz satisface la fórmula. Este algoritmo emplea un enfoque específico para construir el árbol, basado en dicha propiedad. Utiliza un enfoque basado en la generación de un conjunto *lean*[2].

El algoritmo genera un conjunto posible de nodos de tamaño exponencial, con base en el conjunto *lean*. Posteriormente, debe generar diversos candidatos de tipo árbol hasta encontrar alguno que, a partir del nodo raíz se satisfaga la fórmula. En [27] se establece que el tiempo de ejecución del algoritmo es exponencial con respecto al tamaño de su fórmula de entrada. Se realizó la implementación del algoritmo y se experimentó con un conjunto de fórmulas pequeñas.

[2]El conjunto *lean* contiene las partes constitutivas de la fórmula de entrada.

1.2. Contraste con la propuesta

En la Tabla 1.1 se muestra un resumen de algoritmos relacionados con la lógica multimodal K, presentados en la subsección anterior. El algoritmo propuesto en este trabajo, contrasta con los algoritmos existentes por diversas razones.

Se puede observar que el uso de modalidades inversas está presente en [17] y [40], los cuales manejan lógica descriptiva y lógica modal graduada K_R respectivamente. El algoritmo de este trabajo es construido específicamente para la lógica multimodal K tradicional con inversas.

Con base en la información recabada, no se conoce una descripción de un algoritmo que trabaje con inversas para esta lógica, es decir, no hay un diseño nativo. Lo que sugiere que puede tener un mejor rendimiento comparado con un algoritmo diseñado para la lógica descriptiva $\mathcal{ALC}$, la cual es sintácticamente análoga a la lógica multimodal K tradicional. Adicionalmente, no solo se provee un algoritmo sino que además se realiza la implementación respectiva, la cual hasta este momento es la única en su tipo.

El beneficio que proveen las modalidades inversas resalta en aplicaciones que involucran un razonamiento de conceptos como lo es el tiempo, donde es intuitivo que el pasado es la inversa del futuro y viceversa. Esto impacta especialmente en el área de verificación formal, por lo que este trabajo no solamente ofrece una utilidad teórica, sino que también es relevante en aplicaciones de sistemas reales.

Asimismo, en la Tabla 1.1, se puede observar los lenguajes de programación con los que se han implementado los algoritmos, los cuales son bastante diversos. En este trabajo, se realiza la primer implementación de un algoritmo de satisfacción para lógica modal en el lenguaje de programación Go. Este lenguaje es lo suficientemente eficiente para acercarse al rendimiento que ofrece realizar una implementación en C, al mismo tiempo que permite una implementación ágil.

Por otra parte, existen diversos algoritmos que hacen uso de, principalmente, tres técnicas: construcción *Tableau*, traducción a SAT y algoritmos basados en SAT. En este trabajo se emplea un enfoque orientado a la construcción de un modelo con forma de árbol, este enfoque se ha comenzado a explorar en [27] y mediante el uso de autómatas en [34]. El primero para lógica multimodal K y el segundo para lógica modal K, no obstante, nuevamente se resalta la ausencia del manejo de modalidades inversas.

De la misma manera, se proporciona la demostración del teorema de la propiedad de modelo de árbol para la lógica multimodal K con inversas. Esta propiedad permite al algoritmo encontrar un árbol, ante fórmulas que se satisfacen. Además, posibilita acotar el espacio de búsqueda, por ejemplo, al disminuir las redundancias.

Finalmente, la lógica multimodal K no posee restricciones en cuanto a la forma de la es-

tructura de Kripke, por lo que este algoritmo es lo suficientemente general para emplearse de base en el desarrollo de otros algoritmos de satisfacción para otras lógicas modales, ya sea que posean restricciones o sean más expresivas.

Capítulo 2

Lógica multimodal K con inversa

En este capítulo se define la sintaxis y semántica de la lógica multimodal K con inversa, así como los conceptos de satisfacción y validez. Por otra parte, se establece la propiedad de modelo de árbol para esta lógica y se lleva a cabo la demostración del teorema. En todos los casos se presentan ejemplos claros con el objetivo de proporcionar un medio para facilitar la asimilación de los conceptos y definiciones.

El alfabeto de la lógica multimodal K con inversa se compone de un conjunto infinito contable de variables proposicionales P, un conjunto de operadores lógicos $\{\neg, \vee, \wedge, \rightarrow, \leftrightarrow\}$ y de los operadores modales $\Diamond_m$ (posibilidad) y $\Box_m$ (necesidad), tal que m se refiere a una modalidad, dado un conjunto finito de modalidades M.

2.1. Sintaxis

El lenguaje para generar las fórmulas de la lógica multimodal K con inversa se define a continuación:

$$\phi ::= p \mid \neg\,\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \Diamond_m \phi \mid \Box_m \phi$$

donde $p \in P$ y $m \in M$. Una modalidad inversa se representa con una línea horizontal arriba de la modalidad: $\overline{m}$.

Los operadores de la lógica clásica son análogos y se definen como notación de la siguiente manera:

$$\phi \rightarrow \psi \coloneqq \neg\,\phi \vee \psi \qquad \phi \leftrightarrow \psi \coloneqq (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$$

$$\top \coloneqq \phi \vee \neg\phi \qquad \bot \coloneqq \phi \wedge \neg\phi$$

Con el lenguaje definido anteriormente se pueden generar fórmulas sintácticamente válidas para la lógica multimodal K con inversa, a continuación se muestran algunos ejemplos:

$$\Diamond\!\!\!\!\!\!\Diamond_{\boxed{1}}(p \wedge \neg q) \vee \boxed{2}(p \rightarrow q) \qquad r \wedge \Diamond\!\!\!\!\!\!\Diamond_{\boxed{1}}(p \wedge \Diamond\!\!\!\!\!\!\Diamond_{\boxed{1}}\neg q) \qquad \bot \vee \boxed{1}\Diamond\!\!\!\!\!\!\Diamond_{\boxed{2}}(\neg q \leftrightarrow s) \rightarrow \Diamond\!\!\!\!\!\!\Diamond_{\boxed{1}}\Diamond\!\!\!\!\!\!\Diamond_{\boxed{1}}\neg p$$

2.2. Semántica

La estructura de Kripke [22] es la base para interpretar las fórmulas. Se especifica como una tripleta $\mathfrak{M} = (W, \mathcal{R}, V)$, tal que W es un conjunto no vacío de mundos, $\mathcal{R}$ es una familia de relaciones binarias de la forma $R_m \subseteq W \times W$, de manera que m corresponde a una modalidad del conjunto M. Finalmente, V es una función de valuación que se define como $V : P \rightarrow 2^W$. La función $V(p)$ devuelve un subconjunto de mundos de W donde la variable p es verdadera.

Se empleará la notación $[\phi]^{\mathfrak{M}}_w = 1$ para indicar que una fórmula ϕ se sostiene en el mundo $w \in \mathfrak{M}$, otra manera de denotarlo se escribe como $\mathfrak{M}, w \models \phi$ o si se obvia la estructura simplemente $[\phi]_w = 1$. Para indicar lo contrario se escribirá $[\phi]^{\mathfrak{M}}_w = 0$, $[\phi]_w = 0$ o $\mathfrak{M}, w \not\models \phi$. Por otro lado, se utilizará la expresión wR_mw_0 para representar que existe una relación de accesibilidad entre los mundos w y w_0 en el conjunto R_m, es decir, se cumple $(w, w_0) \in R_m$. Finalmente, el conjunto de todas las variables proposicionales de una fórmula ϕ se denotará como $\mathcal{VP}(\phi)$.

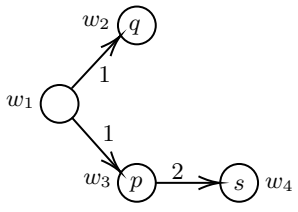
Las reglas para interpretar las fórmulas se definen de manera inductiva [14]. Considere un mundo $w \in W$ que pertenece a una estructura $\mathfrak{M}$, entonces las fórmulas son interpretadas como:

$$\begin{aligned}
[p]_w &= 1, \text{ si y solo si, } w \in V(p).\\
[\neg\,\phi]_w &= 1, \text{ si y solo si, } [\phi]_w = 0.\\
[\phi \wedge \psi]_w &= 1, \text{ si y solo si, } [\phi]_w = 1 \text{ y } [\psi]_w = 1.\\
[\phi \vee \psi]_w &= 1, \text{ si y solo si, } [\phi]_w = 1 \text{ o } [\psi]_w = 1.\\
\left[\Diamond\!\!\!\!\!\!\Diamond_{\boxed{m}}\,\phi\right]_w &= 1, \text{ si y solo si, existe algún } w_0 \in W \text{ con } wR_mw_0 \text{ y } [\phi]_{w_0} = 1.\\
[\boxed{m}\,\phi]_w &= 1, \text{ si y solo si, para cualquier mundo } w_0 \in W\text{: si existe la relación } wR_mw_0,\\
&\qquad\qquad \text{entonces se cumple } [\phi]_{w_0} = 1.
\end{aligned}$$

Definición 1. (Modalidades inversas). En cualquier estructura de Kripke $(W, \mathcal{R}, V)$ se debe cumplir lo siguiente: existe una relación entre dos mundos de W a través de una modalidad m: wR_mw_0, si y solo si, existe la relación inversa mediante su modalidad inversa, es decir, se cumple $w_0R_{\overline{m}}w$. Las relaciones binarias R_m y $R_{\overline{m}}$ están contenidas en la familia de relaciones binarias $\mathcal{R}$.

Considerar modalidades inversas en la lógica multimodal K origina que las estructuras de Kripke deban contener relaciones bidireccionales, debido a que una relación de un mundo a otro mediante una modalidad m, debe permitir regresar al mundo original, mediante la modalidad inversa $\overline{m}$, al ir en la dirección contraría de la relación.

$$\phi_1 = \langle\!\langle 1 \rangle\!\rangle (p \wedge \langle\!\langle 2 \rangle\!\rangle \neg q \wedge [\![2]\!](q \vee s)) \wedge \langle\!\langle 1 \rangle\!\rangle (\neg p \wedge q)$$



$$W = \{w_1,\ w_2,\ w_3,\ w_4\}$$
$$\mathcal{R} = \{R_1, R_2, R_{\overline{1}}, R_{\overline{2}}\}$$
$$R_1 = \{(w_1, w_2), (w_1, w_3)\}$$
$$R_{\overline{1}} = \{(w_2, w_1), (w_3, w_1)\}$$
$$R_2 = \{(w_3, w_4)\} \quad R_{\overline{2}} = \{(w_4, w_3)\}$$
$$V(p) = \{w_3\}$$
$$V(q) = \{w_2\}$$
$$V(s) = \{w_4\}$$

Figura 2.1: Estructura de Kripke $\mathfrak{M}_1$ que satisface ϕ_1.

Ejemplo 2.1. En la Figura 2.1 se muestra un ejemplo de una estructura de Kripke que satisface a la fórmula ϕ_1 desde el mundo w_1. Para comprobarlo, se realizará paso a paso la interpretación de ϕ_1 desde w_1. Primeramente, para que la fórmula se satisfaga, se debe cumplir: $\left[\langle\!\langle 1 \rangle\!\rangle (p \wedge \langle\!\langle 2 \rangle\!\rangle \neg q \wedge [\![2]\!](q \vee s))\right]_{w_1} = 1$ y $\left[\langle\!\langle 1 \rangle\!\rangle (\neg p \wedge q)\right]_{w_1} = 1$.

La interpretación de $\langle\!\langle 1 \rangle\!\rangle (p \wedge \langle\!\langle 2 \rangle\!\rangle \neg q \wedge [\![2]\!](q \vee s))$ indica que debe existir algún mundo adyacente a w_1, a través de la modalidad 1, donde se cumplan las fórmulas: p, $\langle\!\langle 2 \rangle\!\rangle \neg q$ y $[\![2]\!](q \vee s)$. La primera fórmula es una variable proposicional y se cumple en el mundo w_3 debido a que $w_3 \in V(p)$, además que $w_1 R_1 w_3$. La segunda fórmula es una modalidad de posibilidad por lo que debe existir un nodo adyacente a través de la modalidad 2 desde el mundo w_1 donde q no se sostenga. En este caso el mundo w_4 satisface esas condiciones: $[q]_{w_4} = 0$ y $w_3 R_2 w_4$.

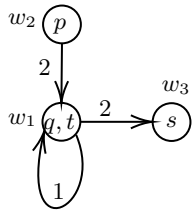
La tercera fórmula posee una modalidad de necesidad, por lo que todos los nodos accesibles desde w_3 mediante la modalidad 2 deben satisfacer la fórmula $q \vee s$. El único mundo accesible con la modalidad 2 es w_4, porque existe la relación $w_3 R_2 w_4$. Como se cumple que $w_4 \in V(s)$, se puede concluir que $[[\![2]\!](q \vee s)]_{w_3} = 1$.

Finalmente, la subfórmula $\langle\!\langle 1 \rangle\!\rangle (\neg p \wedge q)$, ubicada a la derecha del operador de conjunción de ϕ_1, señala que debe existir un mundo adyacente a w_1 mediante la modalidad 1 donde se sostenga la fórmula $\neg p \wedge q$. Un mundo que satisface dichas condiciones es w_2, se puede comprobar fácilmente que $w_2 \notin V(p)$, $w_2 \in V(q)$ y $w_1 R_1 w_2$, por lo que $\left[\langle\!\langle 1 \rangle\!\rangle (\neg p \wedge q)\right]_{w_1} = 1$. En conclusión, la fórmula ϕ_1 se satisface bajo la estructura $\mathfrak{M}_1$ considerando como referencia el mundo w_1.

Ejemplo 2.2. En la Figura 2.2 se muestra la fórmula ϕ_2, la cual contiene modalidades inversas y se satisface bajo la estructura $\mathfrak{M}_2$ desde el mundo w_1. Con el objetivo de corroborarlo, se interpretará la fórmula paso a paso. Es importante recordar que las reglas de interpretación para los operadores modales aplican a cualquier modalidad, incluyendo las inversas.

Con la finalidad de que la fórmula se satisfaga, se deben mantener ambos lados de la conjunción de ϕ_2, es decir, se debe cumplir $\left[\langle\!\langle 2 \rangle\!\rangle (\neg q \wedge p)\right]_{w_1} = 1$ y $\left[\langle\!\langle 1 \rangle\!\rangle (q \wedge [\![\overline{1}]\!](\langle\!\langle 2 \rangle\!\rangle s \wedge [\![\overline{2}]\!](\neg q \vee t)))\right]_{w_1} = 1$. Para el primer caso debe existir un nodo adyacente que esté relacionado desde w_1 me-

$$\phi_2 = \langle\!\langle 2 \rangle\!\rangle(\neg q \wedge p) \wedge \langle\!\langle 1 \rangle\!\rangle(q \wedge [\![\overline{1}]\!](\langle\!\langle 2 \rangle\!\rangle s \wedge [\![\overline{2}]\!](\neg q \vee t)))$$



$$W = \{w_1,\ w_2,\ w_3\} \qquad V(p) = \{w_2\}$$
$$\mathcal{R} = \{R_1, R_{\overline{1}}, R_2, R_{\overline{2}}\} \qquad V(q) = \{w_1\}$$
$$R_1 = \{(w_1, w_1)\} \qquad V(s) = \{w_3\}$$
$$R_{\overline{1}} = \{(w_1, w_1)\} \qquad V(t) = \{w_1\}$$
$$R_2 = \{(w_1, w_3), (w_2, w_1)\} \qquad R_{\overline{2}} = \{(w_3, w_1), (w_1, w_2)\}$$

Figura 2.2: Estructura de Kripke $\mathfrak{M}_2$ con modalidades inversas que satisface ϕ_2.

diante modalidad $\overline{2}$, donde se sostenga la fórmula $\neg q \wedge p$. En el mundo w_2 se mantiene $[\neg q]_{w_2} = 1$ y $[p]_{w_2} = 1$, debido a que $w_2 \notin V(q)$, $w_2 \in V(p)$ y $w_1 R_{\overline{2}} w_2$, por lo tanto $[\neg q \wedge p]_{w_2} = 1$.

La fórmula a la izquierda del operador de conjunción de ϕ_2 corresponde a $\langle\!\langle 1 \rangle\!\rangle(q \wedge [\![\overline{1}]\!](\langle\!\langle 2 \rangle\!\rangle s \wedge [\![\overline{2}]\!](\neg q \vee t)))$ la cual posee un operador de posibilidad, por lo que debe existir al menos un nodo adyacente a w_1, mediante la modalidad 1, donde se cumplan las fórmulas q y $[\![\overline{1}]\!](\langle\!\langle 2 \rangle\!\rangle s \wedge [\![\overline{2}]\!](\neg q \vee t))$. El único mundo donde se sostiene q es en ese mismo nodo w_1, ya que $w_1 \in V(q)$ y existe la relación $w_1 R_1 w_1$.

Posteriormente, se debe verificar que en w_1 se mantenga la segunda fórmula $[\![\overline{1}]\!](\langle\!\langle 2 \rangle\!\rangle s \wedge [\![\overline{2}]\!](\neg q \vee t))$, la cual que posee un operador modal caja. La interpretación indica que todos los mundos accesibles desde w_1 mediante la modalidad $\overline{1}$ deben satisfacer $\langle\!\langle 2 \rangle\!\rangle s \wedge [\![\overline{2}]\!](\neg q \vee t)$. Analizando la estructura de Kripke, solamente hay un único mundo que cumple con dichas características, el cuál es el propio w_1. Por lo que, se debe verificar que se cumpla $\left[\langle\!\langle 2 \rangle\!\rangle s \wedge [\![\overline{2}]\!](\neg q \vee t)\right]_{w_1} = 1$.

Acorde con la interpretación del operador de conjunción, se debe cumplir $\left[\langle\!\langle 2 \rangle\!\rangle s\right]_{w_1} = 1$ y $[[\![\overline{2}]\!](\neg q \vee t)]_{w_1} = 1$. Para el primer caso, el único mundo accesible desde w_1 mediante la modalidad 2 es w_3, y como se cumple que $w_3 \in V(s)$, se concluye que $\left[\langle\!\langle 2 \rangle\!\rangle s\right]_{w_1} = 1$.

Subsecuentemente, en el segundo caso, solamente existe un mundo accesible mediante la modalidad $\overline{2}$ desde w_1, el cual es w_2. Como se sostiene que $w_2 \notin V(q)$, es decir, $[q]_{w_2} = 0$, se concluye que $[\neg q \vee t]_{w_2} = 1$ se cumple y la fórmula ϕ_2 se satisface.

Definición 2. (Forma normal negativa). Una fórmula ϕ se encuentra en forma normal negativa si el operador de negación se aplica únicamente a las variables proposicionales. Para transformarla se define la función `aFNN` que se encarga de recorrer las subfórmulas. Cada vez que se identifique alguna fórmula de la forma $\neg\phi$ se llama a la función `fnn` con

ϕ.

$$\begin{aligned} \texttt{aFNN}(p) &= p & \texttt{aFNN}(\neg\phi) &= \texttt{fnn}(\phi) \\ \texttt{aFNN}(\Diamond_m \phi) &= \Diamond_m \texttt{aFNN}(\phi) & \texttt{aFNN}(\Box_m \phi) &= \Box_m \texttt{aFNN}(\phi) \\ \texttt{aFNN}(\phi \wedge \psi) &= \texttt{aFNN}(\phi) \wedge \texttt{aFNN}(\psi) & \texttt{aFNN}(\phi \vee \psi) &= \texttt{aFNN}(\phi) \vee \texttt{aFNN}(\psi) \end{aligned}$$

La función `fnn` tiene como objetivo enviar el operador de negación a las variables proposicionales, aplicando las leyes de De Morgan, y la notación de la dualidad de los operadores modales $\neg\Diamond_m \phi \coloneqq \Box_m \neg\phi$ y $\neg\Box_m \phi \coloneqq \Diamond_m \neg\phi$, según corresponda.

$$\begin{aligned} \texttt{fnn}(p) &= \neg p & \texttt{fnn}(\neg\phi) &= \texttt{aFNN}(\phi) \\ \texttt{fnn}(\Diamond_m \phi) &= \Box_m \texttt{fnn}(\phi) & \texttt{fnn}(\Box_m \phi) &= \Diamond_m \texttt{fnn}(\phi) \\ \texttt{fnn}(\phi \wedge \psi) &= \texttt{fnn}(\phi) \vee \texttt{fnn}(\psi) & \texttt{fnn}(\phi \vee \psi) &= \texttt{fnn}(\phi) \wedge \texttt{fnn}(\psi) \end{aligned}$$

Ejemplo 2.3. Considere la fórmula ψ_1 presentada en la parte inferior, al transformarla a la forma normal negativa, se observa en ψ_2 que el operador de negación únicamente se aplica a las variables proposicionales.

Fórmula original	Fórmula en forma normal negativa
$\psi_1 = \neg\Diamond_1(p \vee \neg q) \wedge \neg\Box_1\neg\Diamond_2(q \wedge r)$	$\Box_1(\neg p \wedge q) \wedge \Diamond_1\Diamond_2(q \wedge r)$

Satisfacción y Validez.

Con el objetivo de determinar la satisfacción y validez de una fórmula modal se definen los siguientes conceptos:

- **Satisfacción local**. Una fórmula ϕ se satisface localmente, si y solo si, existe alguna estructura $\mathfrak{M} = (W, \mathcal{R}, V)$ que satisface a la fórmula ϕ en algún mundo $w \in W$.

  Ejemplo 2.4. Considere la siguiente fórmula: $\phi_3 = p \wedge \Box_2 q$. Esta fórmula se satisface localmente bajo la estructura $\mathfrak{M}_2$ de la Figura 2.2 debido a que se cumple lo siguiente: $[\phi_3]_{w_1} = 0$, $[\phi_3]_{w_2} = 1$ y $[\phi_3]_{w_3} = 0$. Como se puede observar, ϕ_3 se satisface en al menos un mundo, en este caso únicamente en w_2.

- **Satisfacción global**. Una fórmula ϕ se satisface globalmente, si y solo si, existe alguna estructura $\mathfrak{M} = (W, \mathcal{R}, V)$ que satisface a la fórmula ϕ en cualquier mundo $w \in W$.

  Ejemplo 2.5. Para ilustrar este concepto, considere la fórmula $\phi_4 = \Diamond_1 q \vee \Diamond_1\Diamond_1 p \vee \Diamond_2 p \vee p$ y la estructura $\mathfrak{M}_1$ de la Figura 2.1. Se puede verificar que se sostiene: $[\phi_4]_{w_1} = 1$, $[\phi_4]_{w_2} = 1$, $[\phi_4]_{w_3} = 1$, $[\phi_4]_{w_4} = 1$. Como en todos los mundos de $\mathfrak{M}_1$ la fórmula ϕ_4 se satisface, se concluye que se satisface globalmente.

- **Validez**. Una fórmula ϕ es válida, si y solo si, cualquier estructura $(W, \mathcal{R}, V)$ la satisface globalmente.

Ejemplo 2.6. Con el objetivo de ejemplificar la validez, considere la fórmula $\phi_5 = \boxed{1}(p \to q) \to (\boxed{1}p \to \boxed{1}q)$. Para demostrar que ϕ_5 es válida se realizará una prueba por contradicción. Para ello, suponga que la fórmula no es válida, es decir, que existe al menos una estructura donde ϕ_5 es falsa. Debido a que la fórmula tiene un operador de implicación, para que la fórmula sea falsa el antecedente $\boxed{1}(p \to q)$ debe ser verdadero y el consecuente $\boxed{1}p \to \boxed{1}q$ debe ser falso.

Para que el consecuente sea falso se debe cumplir que $\boxed{1}p$ sea verdadera y $\boxed{1}q$ sea falsa. Es decir, en todos los mundos accesibles mediante la modalidad 1 se debe cumplir p y no se debe cumplir q. Si se considera una estructura con un único mundo que no tiene relaciones mediante la modalidad 1, la formula $\boxed{1}p \to \boxed{1}q$ es verdadera, asi que este caso no conduce a que el consecuente sea falso.

Por otro lado, si se considera alguna estructura con al menos un mundo, que posee relaciones de accesibilidad consigo mismo o hacía otros mundos mediante la modalidad 1 y que además en dichos mundos vecinos se cumple p y no se cumple q, la fórmula $\boxed{1}p \to \boxed{1}q$ es falsa. De esta manera, se han identificado todas las posibles estructuras donde el consecuente de ϕ_5 es falso.

De acuerdo con la hipótesis planteada, la fórmula $\boxed{1}(p \to q)$ debe ser verdadera en las estructuras identificadas del párrafo anterior, donde el consecuente de ϕ_5 es falso. Para que el antecedente de ϕ_5 sea verdadero, la subfórmula $p \to q$ debe ser verdadera. Esto sucede en cualquier caso, excepto cuando p es verdadera y q es falsa. No obstante, en todas las estructuras donde el consecuente de ϕ_5 es falso, y en todos los mundos de dichas estructuras que poseen relaciones de accesibilidad hacía otros mundos o hacía si mismos, mediante la modalidad 1, la fórmula p es verdadera y q falsa, por ende el antecedente es falso en todos estos casos.

En consecuencia, se ha identificado una contradicción debido a que no existe alguna estructura donde ϕ_5 sea falsa. Por lo tanto, queda demostrado que ϕ_5 es válida.

2.3. Propiedad de modelo de árbol

La propiedad de modelo de árbol establece que si existe una estructura de Kripke y un mundo de dicha estructura a partir del cual se satisface una fórmula de lógica modal ϕ, entonces existe un modelo con forma de árbol, que en su nodo raíz, satisface ϕ [14], [42]. Anteriormente, se ha probado que la lógica modal K posee esta propiedad [14]. En este trabajo, se realizará la demostración de la propiedad para la lógica multimodal K con

modalidades inversas.

Esto nos asegura que siempre que una fórmula se satisfaga en alguna estructura, existirá al menos una estructura de Kripke que cumpla la propiedad de modelo de árbol y que también satisfaga la fórmula. Así podemos reducir el espacio de búsqueda, en contraste con una estructura de Kripke arbitraria. Adicionalmente, como no existen ciclos, esto ayuda a construir un algoritmo que termina en tiempo finito.

Sin pérdida de generalidad es posible convertir cualquier fórmula expresada en términos del lenguaje de la lógica multimodal K con inversa a su forma normal negativa, mediante la Definición 2. Por lo tanto, a partir de este punto, todos los teoremas a probar suponen que la fórmula está en esa notación.

Con el propósito de facilitar la prueba, se requiere de una definición auxiliar que permita, a partir de dos estructuras de Kripke, las cuales poseen un nodo donde se sostienen exactamente las mismas variables proposicionales, generar una única estructura mediante la fusión de ese nodo en común.

Definición 3. Dadas dos estructuras de Kripke $\mathfrak{M}' = (W',\ \mathcal{R}',\ V')$, $\mathfrak{M}'' = (W'',\ \mathcal{R}'',\ V'')$, tal que w' y w'' satisfacen las mismas variables proposicionales, dos fórmulas ϕ y ψ con $[\phi]^{\mathfrak{M}'}_{w'} = 1$, $[\psi]^{\mathfrak{M}''}_{w''} = 1$, $w' \in W'$ y $w'' \in W''$, la unión de ambas estructuras $(W,\ \mathcal{R},\ V) = \mathfrak{M}' \cup \mathfrak{M}''$ se define a continuación:
Si $\phi = \psi$, entonces:

- $(W,\ \mathcal{R},\ V) \leftarrow (W',\ \mathcal{R}',\ V')$.

En caso contrario:

- $W \leftarrow (W' \cup W'') \setminus \{w''\}$.
- $\mathcal{R} \leftarrow \mathcal{R}' \cup \mathcal{R}''$.
    - Después, considere las relaciones de accesibilidad $R_m \in \mathcal{R}$:
        - Si $(w'',\ w_0) \in R_m$, entonces se realiza la siguiente actualización: $R_m \leftarrow R_m \cup \{(w',\ w_0)\}$ y $R_m \leftarrow R_m \setminus \{(w'', w_0)\}$.
        - Si $(w_0,\ w'') \in R_m$, entonces se realiza la siguiente actualización: $R_m \leftarrow R_m \cup \{(w_0,\ w')\}$ y $R_m \leftarrow R_m \setminus \{(w_0,\ w'')\}$.

      donde w_0 es cualquier mundo diferente de w' y w'' en W.
- Considere cada variable proposicional $p \in (\mathcal{VP}(\phi) \cup \mathcal{VP}(\psi))$, por cada una se realiza lo siguiente:
    - $V(p) \leftarrow V'(p) \cup V''(p) \setminus \{w''\}$.

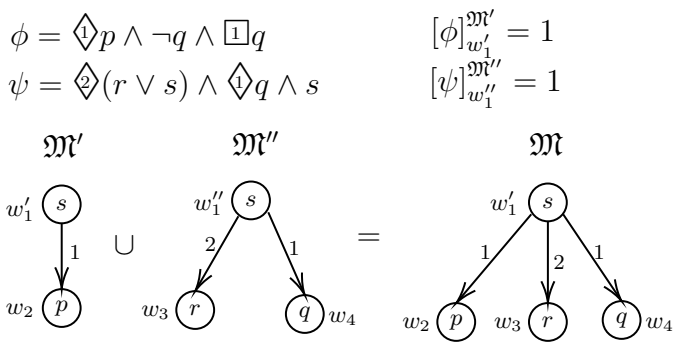


Figura 2.3: Ejemplo de la unión de dos estructuras de Kripke.

Ejemplo 2.7. En la Figura 2.3 se muestra la unión de dos estructuras de Kripke $\mathfrak{M}'$ y $\mathfrak{M}''$ dadas las fórmulas ϕ y ψ que se satisfacen en w'_1 y w''_1 respectivamente. Como resultado se observa que la estructura $\mathfrak{M}$ únicamente conserva uno de los nodos, en este caso w'_1. Esto se debe a que tanto w'_1 como w''_1 satisfacen las mismas variables proposicionales y se toma como punto de referencia para realizar la unión.

La próxima definición genera una copia fresca de una estructura de Kripke. Los mundos son una copia de la estructura original, pero totalmente nuevos. Por lo que las relaciones y función de valuación son idénticas, considerando dichos mundos nuevos. Esta definición se origina cuando en la Definición 5 existe un caso en el que una fórmula con operador modal caja, debido a las modalidades inversas, produce que la subfórmula deba cumplirse en un mundo de nivel superior de la estructura con forma de árbol que se está construyendo. Esta definición permite trasladar la subestructura a dicho nivel superior.

Definición 4. Dada una estructura de Kripke que tiene la propiedad de modelo de árbol $\mathfrak{M}_t = (W, \mathcal{R}, V)$, con nodo raíz $r \in W$ y un conjunto adicional de nodos W_0, es posible crear una estructura que también tiene la propiedad de modelo de árbol $\mathfrak{M}'_t = (W', \mathcal{R}', V')$ y nodo raíz r'. De manera que $\mathfrak{M}'_t$ es una copia de $\mathfrak{M}_t$. Considérese "un mundo nuevo" w' si se cumple que: $w' \notin W$ y $w' \notin W_0$, además que w y w' satisfacen las mismas variables proposicionales.

El proceso para obtener la estructura $\mathfrak{M}'_t$ se detalla a continuación:

- Por cada mundo $w \in W$ se crea una copia w' y dicha copia se añade al conjunto de mundos W', cada nodo w posee una única copia w'.
- Considere las relaciones de accesibilidad $R_m \in \mathcal{R}$:
  - Si $(w, w_0) \in R_m$, entonces se agrega la relación (w', w'_0) a R'_m, tal que w' y w'_0 son las copias frescas generadas de w y w_0, respectivamente.
- Considere todas las variables proposicionales p que se satisfacen en cada mundo $w \in W$:

- Si $w \in V(p)$, entonces se agrega w_f al conjunto $V'(p)$, de manera que w_f es la copia generada de w.

Adicionalmente, para cumplir con el objetivo de probar la propiedad de modelo de árbol, se requiere definir una función que convierta cualquier estructura de Kripke en una estructura con forma de árbol, como se muestra en la siguiente definición.

Definición 5. Se especifica la función $\mathcal{T}$ que recibe como argumentos una fórmula en forma normal negativa ϕ, $\mathfrak{M} = (W, \mathcal{R}, V)$, w, l_r, l_t, ML; tal que $[\phi]_w = 1$, $w \in W$. Los parámetros l_r y l_t son enteros auxiliares que representan el nivel real y relativo durante el recorrido de la estructura original, respectivamente. El parámetro ML es una lista que almacena tuplas del tipo $(l_r^{ML}, l_t^{ML}, m^{ML}, b)$, tal que l_r^{ML}, l_l^{ML} son enteros, b puede tomar el valor de 1 o 0, y m^{ML} es una modalidad. La función retorna una estructura de Kripke que posee la propiedad de modelo de árbol $\mathfrak{M}' = (W', \mathcal{R}', V')$.

Además, se definen los conjuntos $NL = \emptyset$ y $EL = \emptyset$ que serán compartidos con todos los llamados de la misma estructura $\mathfrak{M}$, de manera que $NL \subseteq W$ y los elementos de EL serán tuplas de cuatro elementos $(l_r^{EL}, l_t^{EL}, w^{EL}, \mathfrak{M}'')$ con $w^{EL} \in W''$, $W'' \in \mathfrak{M}''$ y l_r^{EL} y l_t^{EL} valores enteros.

A grandes rasgos, esta función toma una estructura de Kripke que satisface a una fórmula y genera una nueva estructura que cumple la propiedad de modelo de árbol y que satisface la fórmula. Para generarla, se emplean los nodos de la estructura original y copias de estos.

Considere que si un mundo w ya se encuentra en el conjunto NL, entonces en lugar de utilizar w se emplea una copia fresca w_f en la estructura con forma de árbol que se está construyendo. Un mundo w_f es una copia fresca de un mundo $w \in W$ si se cumple que $w_f \notin W$ y ambos mundos satisfacen exacatamente las mismas variables proposicionales.

La función $\mathcal{T}$ se define de manera inductiva a continuación:

- Si $\phi = p$ y $[p]_w^{\mathfrak{M}} = 1$, entonces se agrega w a NL y retorna $\mathfrak{M}' = (W', \{\}, V')$ con $W' = \{w\}$ y $V'(p) = \{w\}$.
- Si $\phi = \neg p$ y $[p]_w^{\mathfrak{M}} = 0$, entonces se incluye w en NL y devuelve $\mathfrak{M}' = (W', \{\}, V')$ con $W' = \{w\}$ y $V'(p) = \{\}$.
- Si $\phi = \psi_1 \vee \psi_2$ y $[\psi_1]_w^{\mathfrak{M}} = 1$, entonces se obtiene $\mathfrak{M}'$ como resultado de llamar a $\mathcal{T}(\psi_1,\ \mathfrak{M},\ w,\ l_r,\ l_t,\ ML)$.
- Si $\phi = \psi_1 \vee \psi_2$ y $[\psi_2]_w^{\mathfrak{M}} = 1$, entonces se retorna $\mathfrak{M}'$ como resultado de invocar a $\mathcal{T}(\psi_2,\ \mathfrak{M},\ w,\ l_r,\ l_t,\ ML)$.
- Si $\phi = \psi_1 \wedge \psi_2$, $[\psi_1]_w^{\mathfrak{M}} = 1$ y $[\psi_2]_w^{\mathfrak{M}} = 1$, entonces retorna la estructura $\mathfrak{M}' \leftarrow \mathcal{T}(\psi_1,\ \mathfrak{M},\ w,\ l_r,\ l_t,\ ML) \cup \mathcal{T}(\psi_2,\ \mathfrak{M},\ w,\ l_r,\ l_t,\ ML)$.

- Si $\phi = \diamondsuit_m \psi$ y existe algún $w_0 \in W$, tal que $(w, w_0) \in R_m$ y $[\psi]^{\mathfrak{M}}_{w_0} = 1$, entonces

  - Si $|ML| > 1$, se examinan las dos últimas tuplas de la lista ML, que serán nombradas como $(l^{ML}_{r1}, l^{ML}_{t1}, m^{ML}_1, b_1)$ y $(l^{ML}_{r2}, l^{ML}_{t2}, m^{ML}_2, b_2)$.
    - Si $l^{ML}_{t1} + 1 = l^{ML}_{t2}$, $b_1 = 0$, $b_2 = 1$ y m^{ML}_1 es la modalidad inversa de m^{ML}_2, se extraen las dos últimas tuplas de la lista ML, se agrega la tupla $(l_r, l_t, m, 0)$ a ML, se obtiene $\mathfrak{M}'_1$ como resultado de invocar a $\mathcal{T}(\psi, \mathfrak{M}, w_0, l_r + 1, l_t + 1, ML)$. Se agrega w a NL, w a W'_1 y (w, w_0) a $R'_{m,1}$ y se crea una copia fresca $\mathfrak{M}''$ de la estructura $\mathfrak{M}'$, acorde a la Definición 4, con nodo raíz $w'' \in W''$. Todos los nodos de W'' se agregan a NL. Se agrega una tupla $(l_t, l^{ML}_{r1}, w'', \mathfrak{M}'')$ al cojunto EL.
    - En caso contrario, se agrega la tupla $(l_r, l_t, m, 0)$ a ML, se obtiene $\mathfrak{M}'_1$ como resultado de invocar a $\mathcal{T}(\psi, \mathfrak{M}, w_0, l_r + 1, l_t + 1, ML)$. Se agrega w a NL, w a W'_1 y (w, w_0) a $R'_{m,1}$.
  - Si $|ML| <= 1$, se agrega la tupla $(l_r, l_t, m, 0)$ a ML, se obtiene $\mathfrak{M}'_1$ como resultado de invocar a $\mathcal{T}(\psi, \mathfrak{M}, w_0, l_r + 1, l_t + 1, ML)$. Se agrega w a NL, w a W'_1 y (w, w_0) a $R'_{m,1}$.

  Finalmente, se analiza cada tupla del conjunto EL, que serán nombradas como $EL_i = ((l^{EL}_{ri}, l^{EL}_{ti}, w^{EL}_i, \mathfrak{M}''^{EL}_i))$. Si $l_r = l^{EL}_{ri}$, $l_t = l^{EL}_{ti}$ y las variables proposicionales de w se sostienen en w^{EL}_i, entonces $\mathfrak{M}' \leftarrow \mathfrak{M}' \cup \mathfrak{M}''^{EL}_i$ y se remueve la tupla EL_i de EL. La función retorna $\mathfrak{M}'$. En caso contrario, simplemente retorna $\mathfrak{M}'$ sin afectar el conjunto EL.

- Si $\phi = \boxed{m}\psi$ y para cualquier mundo $w_0 \in W$, tal que $(w, w_0) \in R_m$ y $[\psi]^{\mathfrak{M}}_{w_0} = 1$, entonces

  - Si $|ML| > 1$, se examinan los dos últimos elementos de la lista ML: $(l^{ML}_{r1}, l^{ML}_{t1}, m^{ML}_1, b_1)$ y $(l^{ML}_{r2}, l^{ML}_{t2}, m^{ML}_2, b_2)$
    - Si $l^{ML}_{t1} + 1 = l^{ML}_{t2}$, $b_1 = 0$, $b_2 = 1$ y m^{ML}_1 es la modalidad inversa de m^{ML}_2, se extraen las dos últimas tuplas mencionadas de la lista ML. Si después de extraerlas se cumple que $|ML| > 0$, se examina la última tupla agregada $(l^{ML}_{r1'}, l^{ML}_{t1'}, m^{ML}_{1'}, b_{1'})$. Se agrega la tupla $(l_r, l_t, m, 1)$ a ML. Si $m^{ML}_{1'}$ es la modalidad inversa de m y $b_{1'} = 0$, entonces se obtiene $\mathfrak{M}'_1$ como resultado de invocar a $\mathcal{T}(\psi, \mathfrak{M}, w_0, l_r + 1, l_t - 1, ML)$, en caso contrario se obtiene $\mathfrak{M}'_1$ como resultado de invocar a $\mathcal{T}(\psi, \mathfrak{M}, w_0, l_r + 1, l_t + 1, ML)$. Se agrega w a NL, w a W'_1 y (w, w_0) a $R'_{m,1}$ y se crea una copia fresca $\mathfrak{M}''$ de la estructura $\mathfrak{M}'$, acorde a la Definición 4, con nodo raíz $w'' \in W''$. Todos los nodos de W'' se agregan a NL. Se agrega una tupla $(l_t, l^{ML}_{r1}, w'', \mathfrak{M}'')$ al conjunto EL.
    - En caso contrario, se agrega la tupla $(l_r, l_t, m, 1)$ a ML, se obtiene $\mathfrak{M}'_1$ como resultado de invocar a $\mathcal{T}(\psi, \mathfrak{M}, w_0, l_r + 1, l_t + 1, ML)$. Se agrega w a NL, w a W'_1 y (w, w_0) a $R'_{m,1}$.

- Si $|ML| = 1$, se examina la tupla $ML = (l_{r1}^{ML}, l_{t1}^{ML}, m_1^{ML}, b_1^{ML})$, se agrega la tupla $(l_r, l_t, m, 1)$ a ML. Si m_1^{ML} es la inversa de m y $b_1^{ML} = 0$, se obtiene $\mathfrak{M}'_1$ como resultado de invocar a $\mathcal{T}(\psi, \mathfrak{M}, w_0, l_r+1, l_t-1, ML)$, en caso contrario, se obtiene $\mathfrak{M}'_1$ como resultado de invocar a $\mathcal{T}(\psi, \mathfrak{M}, w_0, l_r+1, l_t+1, ML)$. Se agrega w a NL, w a W'_1 y (w, w_0) a $R'_{m,1}$.
- En caso contrario, se agrega la tupla $(l_r, l_t, m, 1)$ a ML, se obtiene $\mathfrak{M}'_1$ como resultado de invocar a $\mathcal{T}(\psi, \mathfrak{M}, w_0, l_r+1, l_t+1, ML)$. Se agrega w a NL, w a W'_1 y (w, w_0) a $R'_{m,1}$.

Finalmente, se analiza cada tupla del conjunto EL, que serán nombradas como $EL_i = ((l_{ri}^{EL}, l_{ti}^{EL}, w_i^{EL}, \mathfrak{M}''^{EL}_i))$. Si $l_r = l_{ri}^{EL}$, $l_t = l_{ti}^{EL}$ y las variables proposicionales de w se sostienen en w_i^{EL}, entonces $\mathfrak{M}' \leftarrow \mathfrak{M}' \cup \mathfrak{M}''^{EL}_i$ y se remueve la tupla EL_i de EL, la función retorna $\mathfrak{M}'$. En caso contrario, simplemente retorna $\mathfrak{M}'$ sin afectar EL.

- Si $\phi = \boxed{m}\psi$ y no existe ningún $w_0 \in W$, tal que $(w, w_0) \in R_m$, entonces se incluye w en NL, $W' = \{w\}$ y $\mathfrak{M}' = (W', \{\}, V')$, de manera que, para toda variable proposicional p: $V'(p) = \{\}$.

Teorema 1. Dada una estructura de Kripke $\mathfrak{M} = (W, \mathcal{R}, V)$ y una fórmula ϕ en forma normal negativa, tal que $w \in W$ y $[\phi]_w^{\mathfrak{M}} = 1$, es posible construir una estructura de Kripke $\mathfrak{M}' = (W', \mathcal{R}', V')$ que cumple la propiedad de modelo de árbol, de manera que $w \in W'$ y $[\phi]_w^{\mathfrak{M}'} = 1$, de la siguiente manera.

Demostración

El Teorema 1 se prueba de manera constructiva realizando inducción estructural sobre la fórmula ϕ y auxiliándose de la función $\mathcal{T}$ especificada en la Definición 5.

Casos base

- La fórmula ϕ es una variable proposicional p. Se tiene una estructura $\mathfrak{M}$ que posee un único mundo w donde se cumple ϕ. Es posible construir una estructura $\mathfrak{M}'$ mediante el llamado de la función $\mathcal{T}(\phi, \mathfrak{M}, w, 1, 1, [\,])$. La estructura resultante se define como $\mathfrak{M}' = (\{w\}, \{\}, V')$, con $V'(p) = \{w\}$. Esta estructura satisface ϕ en w, además, un único mundo es una estructura con forma de árbol.
- La fórmula ϕ tiene la forma $\neg p$. Se tiene una estructura $\mathfrak{M}$ que posee un único mundo w donde se satisface ϕ. Mediante el llamado de la función $\mathcal{T}(\phi, \mathfrak{M}, w, 1, 1, [\,])$ se obtiene $\mathfrak{M}'$. La estructura $\mathfrak{M}'$ posee un único mundo donde se cumple ϕ, de manera que $\mathfrak{M}' = (\{w\}, \{\}, V')$, con $V'(p) = \{\}$. Un solo mundo es una estructura con forma de árbol.

Paso inductivo

- La fórmula ϕ tiene la forma $\psi_1 \vee \psi_2$ y se cumple el lado izquierdo de la disyunción. Por inducción se sabe que la función $\mathcal{T}(\psi_1, \mathfrak{M}, w, 1, 1, [\,])$ retorna una estructura con forma de árbol $\mathfrak{M}'$ donde se sostiene ψ_1 en w. Por lo tanto, ϕ también se mantiene en $\mathfrak{M}'$ en el mundo w. El lado izquierdo de la disyunción ψ_2 es un caso análogo al anterior.

- La fórmula tiene la forma $\phi = \psi_1 \wedge \psi_2$. Por inducción se sabe que las dos estructuras generadas por las llamadas a $\mathcal{T}(\psi_1, \mathfrak{M}, w, 1, 1, [\,])$ y $\mathcal{T}(\psi_2, \mathfrak{M}, w, 1, 1, [\,])$ son estructuras con forma de árbol ${\mathfrak{M}'}_{\psi_1}$ y ${\mathfrak{M}'}_{\psi_2}$, que satisfacen a ψ_1 y ψ_2 en w, respectivamente. Como ambas estructuras poseen un nodo raíz w en común, es posible unirlas auxiliándose de la Definición 3, de manera que se crea una nueva estructura $\mathfrak{M}' \leftarrow {\mathfrak{M}'}_{\psi_1} \cup {\mathfrak{M}'}_{\psi_2}$. La estructura $\mathfrak{M}'$ mantiene la forma de árbol porque ${\mathfrak{M}'}_{\psi_1}$ y ${\mathfrak{M}'}_{\psi_2}$ tienen forma de árbol y la unión se hace sobre un nodo raíz en común.

  Adicionalmente, se mantiene la satisfacción de ψ_1 y ψ_2 bajo $\mathfrak{M}'$ en w, debido a que los nodos vecinos de w en $\mathfrak{M}'$ son los mismos que los de la estructura original $\mathfrak{M}$ y a lo más copias de dichos vecinos, además que no se modifican las variables proposicionales que se cumplen en dichos nodos, ni en sus copias. De esta manera, después de la unión, como ψ_1 y ψ_2 se satisfacen en el mundo w de $\mathfrak{M}'$, también se satisface ϕ.

- La fórmula ϕ posee la forma $\diamondsuit_{\boxed{m}}\psi$. En la estructura $\mathfrak{M} = (W, \mathcal{R}, V)$ se sostiene ϕ en el mundo $w \in W$. Existe al menos un nodo $w_0 \in W$ donde se mantiene ψ, de manera que $(w, w_0) \in R_m$. Se realiza una llamada a $\mathcal{T}(\psi, \mathfrak{M}, w_0, 1, 1, [\,])$ que por inducción devuelve una estructura con forma de árbol $\mathfrak{M}'_\psi = (W'_\psi, R'_\psi, V'_\psi)$ que satisface a ψ en el nodo raíz w'_0 de W'_ψ.

  Se puede construir una estructura con forma de árbol $\mathfrak{M}' = (W', \mathcal{R}', V')$ si se agrega w al conjunto W', (w, w'_0) a R'_m y (w'_0, w) en $R'_{\overline{m}}$. Además, se incluyen todos los mundos de W'_ψ en W', así como $\mathcal{R}' = \mathcal{R}' \cup R'_\psi$ y la función de valuación se actualiza considerando todas las variables proposicionales p en ψ obteniendo $V'(p) \leftarrow V'(p) \cup V'_\psi(p)$. La estructura resultante $\mathfrak{M}'$ posee forma de árbol, puesto que las relaciones existentes entre w y w'_0 mantienen esta forma. Además, es evidente que en el mundo w de W' se cumple ϕ.

- La fórmula tiene la forma: $\phi = \boxed{m}\psi$.

  - La estructura $\mathfrak{M}$ posee un único mundo w donde se cumple ϕ. No existe ningún nodo $w_0 \in W$ de manera que se cumpla $(w, w_0) \in R_m$. La estructura con forma de árbol se construye de la siguiente manera: $\mathfrak{M}' = (\{w\}, \{\,\}, V')$, considerando que si $w \in V(p)$, entonces $V'(p) = \{w\}$, tal que p son las variables proposicionales de ψ.
  - La estructura $\mathfrak{M} = (W, \mathcal{R}, V)$ posee al menos un mundo $w \in W$ donde se satisface ϕ y desde ese mundo existe al menos una relación de accesibilidad mediante la modalidad m hacia cualquier mundo. Por cada relación $(w, w_i) \in R_m$,

se puede obtener por hipótesis de inducción una estructura con forma de árbol $\mathfrak{M}'_i = (V'_i,\ \mathcal{R}'_i,\ V'_i)$ al llamar a la función $\mathcal{T}(\psi, \mathfrak{M}, w_i, 1, 1, [\,])$, que satisface a ψ.

Posteriormente, se construye otra estructura con forma de árbol $\mathfrak{M}' = (W',\ \mathcal{R}',\ V')$ de manera que (w, w'_i) está en el conjunto R'_m y (w_i, w) en $R'_{\overline{m}}$. Adicionalmente, por cada estructura $\mathfrak{M}'_i$, todos los mundos de W'_i están presentes en W', así como $\mathcal{R}' \leftarrow \mathcal{R}' \cup R'_i$ y por cada variable proposicional p en ψ se sostiene $V'(p) \leftarrow V'(p) \cup V'_i(p)$. Es claro que la estructura $\mathfrak{M}'$ posee forma de árbol. En el nodo w de $\mathfrak{M}'$ se sostiene ϕ.

Por lo tanto, queda demostrado el Teorema 1.

Ejemplo 2.8. Empleando la Definición 5, se realizará la transformación de una estructura de Kripke en una estructura de tipo árbol para mostrar el resultado de una conversión. Se considera $\mathfrak{M}_2$ y la fórmula $\phi_2 = \Diamond_{\boxed{2}}(\neg q \wedge p) \wedge \Diamond_{\boxed{1}}(q \wedge \boxed{\overline{1}}(\Diamond_{\boxed{2}} s \wedge \boxed{\overline{2}}(\neg q \vee t)))$ mostrada en la Figura 2.2, que se satisface en el mundo w_1 bajo dicha estructura. Se empleará un subíndice numérico, por ejemplo R_{2_m}, para hacer referencia a todas las relaciones de accesibilidad de la modadlidad m asociadas, en este caso, a la familia de relaciones $\mathcal{R}_2$ de la estructura $\mathfrak{M}_2$.

1. $\mathcal{T}(\phi_2,\ \mathfrak{M}_2,\ w_1,\ l_r = 1,\ l_t = 1,\ ML = [\,])$, $NL = \{\,\}$ y $EL = \{\,\}$. La fórmula tiene la forma de una conjunción, por lo que entra en el caso $\phi_2 = \psi_1 \wedge \psi_2$. La función retorna la estructura $\mathfrak{M}'_1 \leftarrow \mathcal{T}(\psi_1,\ \mathfrak{M}_2, w_1,\ l_r,\ l_t,\ ML) \cup \mathcal{T}(\psi_2,\ \mathfrak{M}_2, w_1,\ l_r,\ l_t,\ ML)$, tal que $\psi_1 = \Diamond_{\boxed{2}}(\neg q \wedge p)$ y $\psi_2 = \Diamond_{\boxed{1}}(q \wedge \boxed{\overline{1}}(\Diamond_{\boxed{2}} s \wedge \boxed{\overline{2}}(\neg q \vee t)))$.

2. $\mathcal{T}(\psi_1,\ \mathfrak{M}_2,\ w_1,\ l_r = 1,\ l_t = 1,\ ML = [\,])$, $NL = \{\,\}$ y $EL = \{\,\}$. La fórmula es modal y corresponde al caso $\psi_1 = \Diamond_{\boxed{2}}\psi_3$, tal que $\psi_3 = \neg q \wedge p$. En el mundo $w_2 \in \mathfrak{M}_2$ se sostiene ψ_3, además existe la relación $w_1 R_{2_{\overline{2}}} w_2$. Se agrega el mundo w_1 al conjunto NL. Se agrega la tupla $(l_r,\ l_t,\ \overline{2},\ 0)$ a ML. Se realiza el llamado $\mathfrak{M}'_2 \leftarrow \mathcal{T}(\psi_3,\ \mathfrak{M}_2, w_2,\ l_r + 1,\ l_t + 1,\ ML = [(1,\ 1,\ \overline{2},\ 0)])$, de manera que w_1 se incluye en el conjunto W'_2, la relación (w_1, w_2) en $R'_{2_{\overline{2}}}$, (w_2, w_1) en R'_{2_2}, $w_1 \in V'_2(q)$ y $w_1 \in V'_2(t)$. En este caso w_2 es el nuevo nodo raíz de $\mathfrak{M}'_2$. La función devuelve la estructura $\mathfrak{M}'_2$.

3. $\mathcal{T}(\psi_3,\ \mathfrak{M}_2,\ w_2,\ l_r = 2,\ l_t = 2,\ ML = [(1,\ 1,\ \overline{2},\ 0)])$, $NL = \{w_1\}$ y $EL = \{\,\}$. La fórmula contiene una conjunción y entra al caso $\psi_3 = \psi_4 \wedge \psi_5$, tal que $\psi_4 = \neg q$ y $\psi_5 = p$. Retorna una estructura $\mathfrak{M}'_3 \leftarrow \mathcal{T}(\psi_4,\ \mathfrak{M}_2, w_2,\ l_r,\ l_t,\ ML) \cup \mathcal{T}(\psi_5,\ \mathfrak{M}_2, w_2,\ l_r,\ l_t,\ ML)$.

4. $\mathcal{T}(\psi_4,\ \mathfrak{M}_2,\ w_2,\ l_r = 2,\ l_t = 2,\ ML = [(1,\ 1,\ \overline{2},\ 0)])$, $NL = \{w_1\}$ y $EL = \{\,\}$. La fórmula es la negación de una variable proposicional, $\psi_4 = \neg q$. Se agrega w_2 al conjunto NL. Este llamado retorna $\mathfrak{M}'_4 = (\{w_2\},\ \{\,\},\ V'_4)$, con $V'_4(p) = \{w_2\}$.

5. $\mathcal{T}(\psi_5,\ \mathfrak{M}_2,\ w_2,\ l_r = 2,\ l_t = 2,\ ML = [(1,\ 1,\ \overline{2},\ 0)])$, $NL = \{w_1, w_2\}$ y $EL = \{\,\}$. La fórmula es una variable proposicional. Como w_2 ya está presente en NL, se crea una copia fresca w'_2 y se agrega a NL. Entonces, la función retorna $\mathfrak{M}'_5 = (\{w'_2\},\ \{\,\},\ V'_5)$, con $V'_5(p) = \{w'_2\}$.

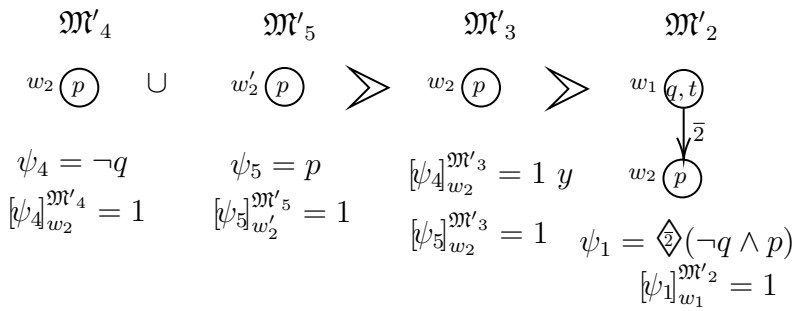


Figura 2.4: Generación de las estructuras con forma de árbol $\mathfrak{M}'_3$, que satisface ψ_4 y ψ_5 en w_2, mediante la unión de $\mathfrak{M}'_4$ y $\mathfrak{M}'_5$; y $\mathfrak{M}'_2$ que satisface ψ_1 en w_1, del paso 2.

6. *a*) Se obtiene $\mathfrak{M}'_3$ mediante $\mathfrak{M}'_4 \cup \mathfrak{M}'_5$. La unión de ambas estructuras mediante el uso de la Definición 3 resulta en $\mathfrak{M}'_3 = (\{w_2\}, \{\}, V'_3)$, con $V'_3(p) = \{w_2\}$, como se visualiza en la Figura 2.4.

   b) La estructura $\mathfrak{M}'_2$ se visualiza en la Figura 2.4 y se construye como se especifica en el Paso 2, la cual resulta en la tripleta $(W'_2, \mathcal{R}'_2, V'_2)$, tal que $W'_2 = \{w_1, w_2\}$, $(w_1, w_2) \in R'_{2_{\overline{2}}}$, $(w_2, w_1) \in R'_{2_2}$, $V'_2(p) = \{w_2\}, V'_2(q) = \{w_1\}$ y $V'_2(t) = \{w_1\}$.

7. $\mathcal{T}(\psi_2, \mathfrak{M}_2, w_1, l_r = 1, l_t = 1, ML = [\,])$, $NL = \{w_1, w_2, w'_2\}$ y $EL = \{\,\}$. Corresponde a un caso modal de posibilidad, de manera que $\psi_2 = \langle\!\langle 1 \rangle\!\rangle \psi_6$, con $\psi_6 = q \wedge [\![\overline{1}]\!](\langle\!\langle 2 \rangle\!\rangle s \wedge [\![\overline{2}]\!](\neg q \vee t))$. En el mundo $w_1 \in \mathfrak{M}_2$ se sostiene ψ_6, además se cumple que $w_1 R_{2_1} w_1$. Entonces, como ya existe w_1 en NL se crea una copia fresca w'_1 y se agrega a NL. Se agrega la tupla $(l_r, l_t, 1, 0)$ a ML. Se realiza el llamado $\mathfrak{M}'_6 \leftarrow \mathcal{T}(\psi_6, \mathfrak{M}_2, w_1, l_r + 1, l_t + 1, ML = [(1, 1, 1, 0)])$ y se agrega w'_1 a W'_6, además se relaciona el nodo raíz $r_1 \in W'_6$ con el nuevo nodo raíz w'_1, de manera que existen las relaciones (w'_1, r_1) en R'_{6_1} y (r_1, w'_1) en $R'_{6_{\overline{1}}}$. Adicionalmente $w'_1 \in V'_2(q)$ y $w'_1 \in V'_2(t)$.

   La llamada a la función $\mathcal{T}$ provoca una actualización en el conjunto EL, de manera que $EL = \{(1, 1, w_1^V, \mathfrak{M}''_{11}), (1, 1, w_1^{VI}, \mathfrak{M}''_{13})\}$. Posteriormente, se analiza cada tupla del conjunto EL, las cuales serán llamadas $EL_i = ((l_{ri}^{EL}, l_{ti}^{EL}, w_i^{EL}, \mathfrak{M}''^{EL}_i))$. Si $l_r = l_{ri}^{EL}$, $l_t = l_{ti}^{EL}$ y las variables proposicionales de w'_1 se sostienen en w_i^{EL}, entonces $\mathfrak{M}'_6 \leftarrow \mathfrak{M}'_6 \cup \mathfrak{M}''^{EL}_i$ y se remueve la tupla EL_i de EL, la función retorna $\mathfrak{M}'_6$.

8. $\mathcal{T}(\psi_6, \mathfrak{M}_2, w_1, l_r = 2, l_t = 2, ML = [(1, 1, 1, 0)])$, $NL = \{w_1, w_2, w'_2, w'_1\}$ y $EL = \{\,\}$. La fórmula posee un operador de conjunción: $\psi_6 = \psi_7 \wedge \psi_8$, tal que $\psi_7 = q$ y $\psi_8 = [\![\overline{1}]\!](\langle\!\langle 2 \rangle\!\rangle s \wedge [\![\overline{2}]\!](\neg q \vee t))$. La función devuelve la estructura $\mathfrak{M}'_7$ compuesta por $\mathcal{T}(\psi_7, \mathfrak{M}_2, w_1, l_r, l_t, ML) \cup \mathcal{T}(\psi_8, \mathfrak{M}_2, w_1, l_r, l_t, ML)$.

9. $\mathcal{T}(\psi_7, \mathfrak{M}_2, w_1, l_r = 2, l_t = 2, ML = [(1, 1, 1, 0)])$, $NL = \{w_1, w_2, w'_2, w'_1\}$ y $EL = \{\,\}$. Corresponde a un caso de variable proposicional $\psi_7 = q$. Los mundos w_1 y w'_1 ya

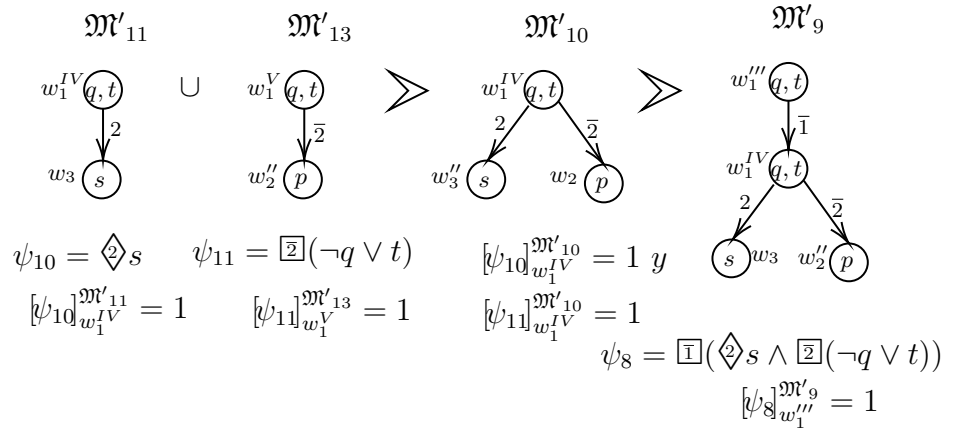


Figura 2.5: Estructura $\mathfrak{M}'_{10}$ generada del paso 11 y estructura $\mathfrak{M}'_{9}$ creada en el paso 10.

están en NL, por lo tanto se crea una copia fresca w_1'' y se agrega a NL. La función retorna la estructura $\mathfrak{M}'_8 \leftarrow (\{w_1''\}, \{\}, V'_8)$, con $V'_8(q) = \{w_1''\}$, $V'_8(t) = \{w_1''\}$.

10. $\mathcal{T}(\psi_8, \mathfrak{M}_2, w_1, l_r = 2, l_t = 2, ML = [(1, 1, 1, 0)])$, $NL = \{w_1, w_2, w_2', w_1', w_1''\}$ y $EL = \{\}$. La fórmula posee un operador modal de necesidad: $\psi_8 = \boxed{\overline{1}}\psi_9$, con $\psi_9 = \Diamond_{\boxed{2}} s \wedge \boxed{\overline{2}}(\neg q \vee t)$. A partir del mundo w_1 únicamente existe una relación, mediante la modalidad 1, con el mundo w_1 y en w_1 se cumple ψ_9. Debido a que w_1, w_1' y w_1'' ya están en NL, se crea una copia fresca w_1''' y se agrega al conjunto NL. Se agrega la tupla $(2, 2, \overline{1}, 1)$ a ML. Se crea la estructura $\mathfrak{M}'_9 \leftarrow \mathcal{T}(\psi_9, \mathfrak{M}_2, w_1, l_r + 1, l_t - 1, ML = [(1, 1, 1, 0), (2, 2, \overline{1}, 1)])$, su nodo raíz es r'_1. En la estructura $\mathfrak{M}'_9$ se agrega el nuevo nodo raíz w_1'''; las relaciones $(w_1''', r'_1) \in R'_{9_1}$ y $(r'_1, w_1''') \in R'_{9_{\overline{1}}}$; la función de valuación posee $w_1''' \in V'_9(q)$ y $w_1''' \in V'_9(t)$. Se retorna $\mathfrak{M}'_9$

    Nota: Como es un operador de necesidad, si existiera más de un mundo w_0 de manera que se cumpla $w_1 R_m w_0$ y se satisfaga ψ_9, como se menciona en la Definición 5, se tendría que relacionar el nuevo nodo raíz w_1''' con cada nodo raíz de la estructura que genere cada llamado a la función $\mathcal{T}$. Por otra parte, si no hay mundos, no se realiza ningún llamado y retorna directamente la estructura con un único nodo donde se cumple ψ_8, en este caso.

11. $\mathcal{T}(\psi_9, \mathfrak{M}_2, w_1, l_r = 3, l_t = 1, ML = [(1, 1, 1, 0), (2, 2, \overline{1}, 1)])$, $NL = \{w_1, w_2, w_2', w_1', w_1'', w_1'''\}$ y $EL = \{\}$. El operador inmediato de la fórmula es una conjunción: $\psi_9 = \psi_{10} \wedge \psi_{11}$ con $\psi_{10} = \Diamond_{\boxed{2}} s$ y $\psi_{11} = \boxed{\overline{2}}(\neg q \vee t)$. La función retorna la estructura $\mathfrak{M}'_{10} \leftarrow \mathcal{T}(\psi_{10}, \mathfrak{M}_2, w_1, l_r, l_t, ML) \cup \mathcal{T}(\psi_{11}, \mathfrak{M}_2, w_1, l_r, l_t, ML)$.

12. $\mathcal{T}(\psi_{10}, \mathfrak{M}_2, w_1, l_r = 3, l_t = 1, ML = [(1, 1, 1, 0), (2, 2, \overline{1}, 1)])$, $NL = \{w_1, w_2, w_2', w_1', w_1'', w_1'''\}$ y $EL = \{\}$. La fórmula posee un operador modal: $\psi_{10} = \Diamond_{\boxed{2}} \psi_{12}$ con $\psi_{12} = s$. Existe un mundo $w_3 \in \mathfrak{M}_2$ de manera que $w_1 R_{2_2} w_3$ y se cumple ψ_{12}. Se genera un nuevo nodo fresco w_1^{IV} y se agrega a NL. Como $|ML| > 1$ se examinan las últimas dos tuplas $(l_{r1}^{ML}, l_{t1}^{ML}, m_1^{ML}, b_1)$ y $(l_{r2}^{ML}, l_{t2}^{ML}, m_2^{ML}, b_2)$, como se cumple que $l_{t1}^{ML} + 1 = l_{t2}^{ML}$, $b_1 = 0$, $b_2 = 1$ y m_1^{ML} es la modalidad inversa de m_2^{ML}, entonces

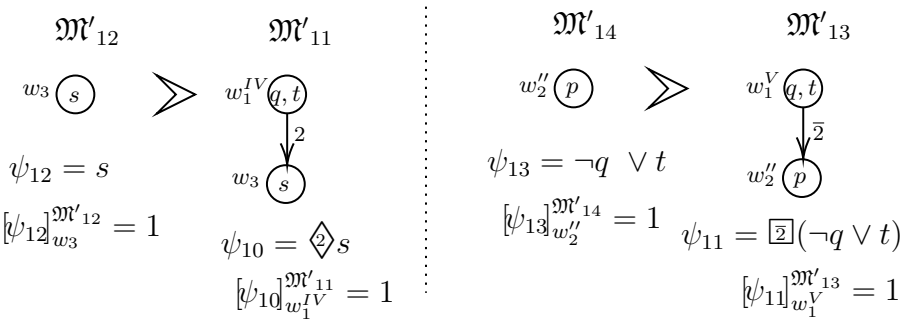


Figura 2.6: Generación de la estructura con forma de árbol $\mathfrak{M}'_{11}$, que satisface ψ_{10} en w_1^{IV}; y $\mathfrak{M}'_{13}$, que satisface ψ_{11} en w_1^{V}. Obtenidas de los pasos 12 y 15 respectivamente.

se extraen las dos últimas tuplas mencionadas de la lista ML, se agrega la tupla $(l_r,\, l_t,\, 2,\, 0)$ a ML y se efectúa un llamado para obtener $\mathfrak{M}'_{11} \leftarrow \mathcal{T}(\psi_{12},\, \mathfrak{M}_2,\, w_3,\, l_r+1,\, l_t+1,\, ML = [(3,\, 1,\, 2,\, 0)])$, su nodo raíz se etiquetará como r_3.

Se agrega w_1^{IV} al conjunto W'_{11}; las relaciones binarias: $(w_1^{IV},\, r_3)$ a R'_{11_2} y $(r_3,\, w_1^{IV})$ a $R'_{11_{\overline{2}}}$; y en la función de valuación se considera $w_3 \in V(s)$. Finalmente, se crea una copia fresca $\mathfrak{M}''_{11}$ de la estructura $\mathfrak{M}'_{11}$, acorde a la Definición 4, con nodo raíz $w_1^{V} \in W''$. Todos los nodos de W'' se agregan a NL. Se agrega una tupla $(l_t = 1,\, l_{r1}^{ML} = 1,\, w_1^{V},\, \mathfrak{M}''_{11})$ al conjunto EL. Se devuelve $\mathfrak{M}'_{11}$.

13. $\mathcal{T}(\psi_{12},\, \mathfrak{M}_2,\, w_3,\, l_r = 4,\, l_t = 2,\, ML = [(3,\, 1,\, 2,\, 0)])$, $NL = \{w_1, w_2, w_2', w_1', w_1'', w_1''', w_1^{IV}, w_1^{V}, w_2''', w_3'\}$ y $EL = \{(1,\, 1,\, w_1^{V},\, \mathfrak{M}''_{11})\}$. La fórmula es una variable proposicional $\psi_{12} = s$, por lo que se agrega w_3 a NL y la función devuelve la estructura: $\mathfrak{M}'_{12} = (\{w_3\}, \{\,\}, V'_{12})$, con $V'_{12}(s) = \{w_3\}$.

14. *a*) Se construye $\mathfrak{M}'_{11}$ como se especifica en el paso 12. Lo que resulta en la siguiente estructura: $(\{w_1^{IV}, w_3\};\, (w_1^{IV},\, w_3) \in R'_{11_2},\, (w_3,\, w_1^{IV}) \in R'_{11_{\overline{2}}};\, V'_{11}(q) = \{w_1^{IV}\},\, V'_{11}(t) = \{w_1^{IV}\},\, V'_{11}(s) = \{w_3\})$. Se visualiza de manera gráfica en la Figura 2.6.

15. $\mathcal{T}(\psi_{11},\, \mathfrak{M}_2,\, w_1,\, l_r = 3,\, l_t = 1,\, ML = [(1,\, 1,\, 1,\, 0),\, (2,\, 2,\, \overline{1},\, 1)])$, $NL = \{w_1, w_2, w_2', w_1', w_1'', w_1''', w_1^{IV}, w_3, w_1^{V}, w_2''', w_3'\}$ y $EL = \{(1,\, 1,\, w_1^{V},\, \mathfrak{M}''_{11})\}$. La fórmula corresponde al caso del operador modal de necesidad: $\psi_{11} = [\![\overline{2}]\!]\psi_{13}$, tal que $\psi_{13} = \neg q \vee t$. El mundo a considerar es w_2, ya que es el único que posee una relación $w_1 R_{2_{\overline{2}}} w_2$ en $\mathfrak{M}_2$ con la modalidad requerida y en w_2 se mantiene ψ_{13}. Se crea una copia fresca w_1^{V} y se agrega a NL. Como $|ML| > 1$ se examinan las últimas dos tuplas $(l_{r1}^{ML},\, l_{t1}^{ML}, m_1^{ML}, b_1)$ y $(l_{r2}^{ML},\, l_{t2}^{ML},\, m_2^{ML}, b_2)$, como se cumple que $l_{t1}^{ML} + 1 = l_{t2}^{ML}$, $b_1 = 0$, $b_2 = 1$ y m_1^{ML} es la modalidad inversa de m_2^{ML}, entonces se extraen las dos últimas tuplas mencionadas de la lista ML.

Como no se cumple $|ML| > 0$ después de extraerlas se procede a agregar la tupla $(l_r,\, l_t,\, 2,\, 1)$ a ML, se efectúa un llamado para obtener $\mathfrak{M}'_{13} = \mathcal{T}(\psi_{13},\, \mathfrak{M}_2,\, w_2,\, l_r + 1,\, l_t + 1,\, ML = [(3,\, 1,\, \overline{2},\, 1)])$ de manera que se agrega w_1^{V} a W'_{13}; las relaciones

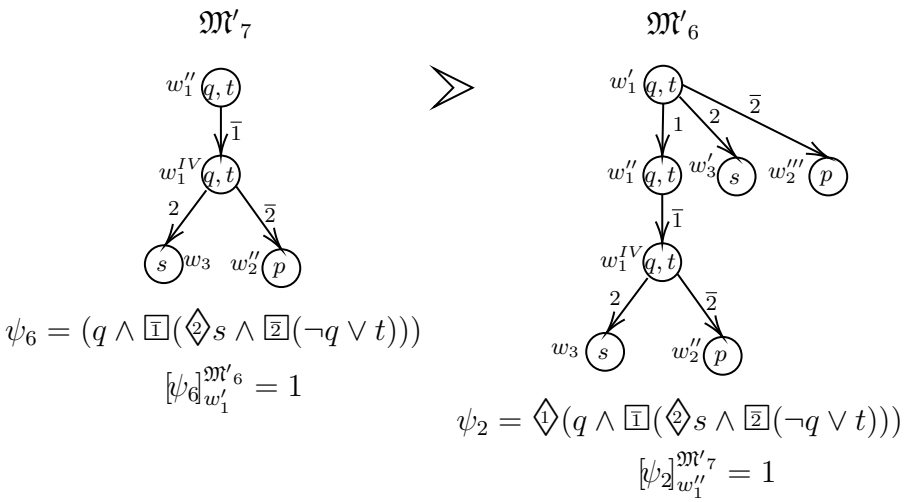


Figura 2.7: Creación de la estructura $\mathfrak{M}'_6$ que satisface a ψ_2 en el nodo w'_1 obtenida en el paso 7.

binarias (w_1^V, r_2) se agregan a $R'_{13_{\overline{2}}}$ y (r_2, w_1^V) a R'_{13_2}; finalmente, se incluye w_1^V en $V'_{13}(q)$ y w_1^V en $V'_{13}(t)$, con el objetivo de unir el nuevo nodo raíz w_1^V a la estructura $\mathfrak{M}'_{13}$. Se crea una copia fresca $\mathfrak{M}''_{13}$ de la estructura $\mathfrak{M}'_{13}$, acorde a la Definición 4, con nodo raíz $w_1^{VI} \in W''$. Todos los nodos de W''_{13} se agregan a NL. Se agrega una tupla ($l_t = 1$, $l_{r1}^{ML} = 1$, w_1^{VI}, $\mathfrak{M}''_{13}$) al conjunto EL. Se retorna $\mathfrak{M}'_{13}$.

16. $\mathcal{T}(\psi_{13}, \mathfrak{M}_2, w_2, l_4 = 4, l_t = 2, ML = [(3, 1, \overline{2}, 1)])$, $NL = \{w_1, w_2, w'_2, w'_1, w''_1, w'''_1, w^{IV}_1, w_3, w^V_1, w'''_2, w'_3, w^{IV}_2, w''_3\}$ y $EL = \{(1, 1, w_1^V, \mathfrak{M}''_{11}), (1, 1, w_1^{VI}, \mathfrak{M}''_{13})\}$. El operador inmediato es una disyunción de la forma: $\psi_{13} = \psi_{14} \vee \psi_{15}$ con $\psi_{14} = \neg q$ y $\psi_{15} = t$. En el mundo recibido por el tercer parámetro $w_2 \in \mathfrak{M}_2$ de este llamado se cumple ψ_{14}. Por lo que la función retorna la estructura $\mathfrak{M}'_{14} \leftarrow \mathcal{T}(\psi_{14}, \mathfrak{M}_2, w_2, l_r, l_t, ML)$.

17. $\mathcal{T}(\psi_{14}, \mathfrak{M}_2, w_2, l_4 = 4, l_t = 2, ML = [(3, 1, \overline{2}, 1)])$, $NL = \{w_1, w_2, w'_2, w'_1, w''_1, w'''_1, w^{IV}_1, w_3, w^V_1, w'''_2, w'_3, w^{IV}_2, w''_3\}$ y $EL = \{(1, 1, w_1^V, \mathfrak{M}''_{11}), (1, 1, w_1^{VI}, \mathfrak{M}''_{13})\}$. La fórmula es la negación de una variable proposicional con $\psi_{14} = \neg q$. Se crea un mundo fresco w''_2 y se agrega a NL. La función devuelve $\mathfrak{M}'_{15} \leftarrow (\{w''_2\}, \{\}, V'_{15})$, con $V'_{15} = \{p\}$.

18. *a*) Se genera $\mathfrak{M}'_{13}$ como se especifica en el paso 15. El resultado es una estructura $(W'_{13}, \mathcal{R}'_{13}, V'_{13})$, tal que $\{w_1^V, w''_2\} \in W'_{13}$; $(w_1^V, w''_2) \in R'_{13_{\overline{2}}}$, $(w''_2, w_1^V) \in R'_{13_2}$; $V'_{13}(q) = \{w_1^V\}$, $W'_{13}(t) = \{w_1^V\}$ y $V'_{13}(p) = \{w''_2\}$. Esta se muestra en la Figura 2.6.

    b) Se construye $\mathfrak{M}'_{10}$ detallado en el paso 11, la estructura es $(W'_{10}, \mathcal{R}'_{10}, V'_{10})$, con $\{w_1^{IV}, w_3, w''_2\} \in W'_{10}$; $\{(w_1^{IV}, w_3), (w''_2, w_1^{IV})\} \in R'_{10_2}$, $\{(w_3, w_1^{IV}), (w_1^{IV}, w''_2)\} \in R'_{10_{\overline{2}}}$; $V'_{10}(q) = \{w_1^{IV}\}$, $V'_{10}(t) = \{w_1^{IV}\}$, $V'_{10}(s) = \{w_3\}$ y $V'_{10}(p) = \{w''_2\}$. Se puede observar en la Figura 2.5.

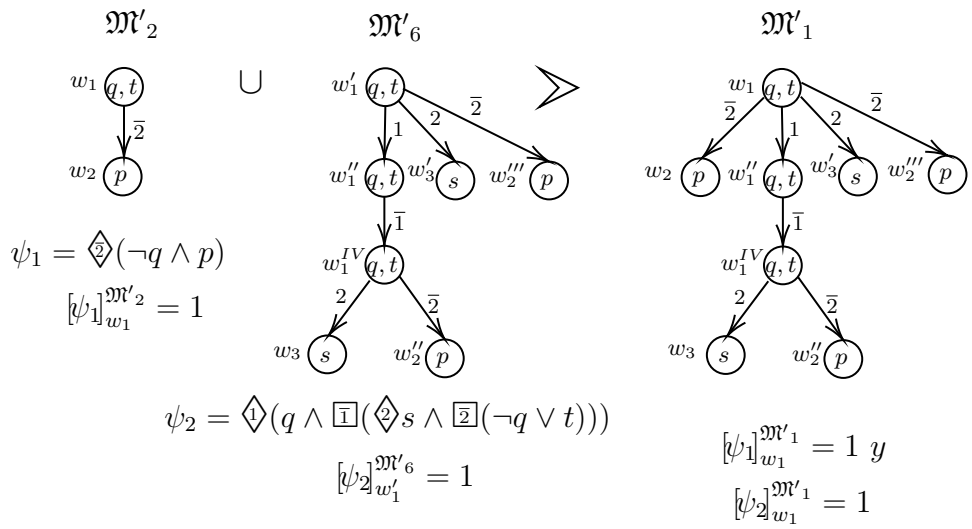


Figura 2.8: Creación de la estructura final con forma de árbol $\mathfrak{M}'_1$, que satisface la fórmula original ϕ_2 en el nodo raíz w_1, obtenida en el paso 1.

c) Se establece $\mathfrak{M}'_9$ como se indica en el paso 10. La estructura resultante es $(W'_9, \mathcal{R}'_9, V'_9)$, tal que $\{w_1^{IV}, w_3, w''_2, w'''_1\} \in W'_9$; $(w'''_1,\, w_1^{IV}) \in R'_{9_{\bar{1}}}$, $(w_1^{IV},\, w'''_1) \in R'_{9_1}$, $\{(w_1^{IV}, w_3),\, (w''_2, w_1^{IV})\} \in R'_{9_2}$, $\{(w_3, w_1^{IV}),\, (w_1^{IV}, w''_2)\} \in R'_{9_{\bar{2}}}$; $V'_9(q) = \{w_1^{IV},\, w'''_1\}$, $V'_9(t) = \{w_1^{IV},\, w'''_1\}$, $V'_9(s) = \{w_3\}$ y $V'_9(p) = \{w''_2\}$. Se aprecia gráficamente en la Figura 2.5.

d) Se genera $\mathfrak{M}'_7$ como se especifica en el paso 8. Este paso no se detallará debido a que la estructura no cambia, simplemente el nodo raíz w'''_1 se renombra a w''_1. La estructura de Kripke de manera gráfica se observa en la Figura 2.7.

e) Se construye $\mathfrak{M}'_6$ detallado en el paso 7. La estructura se define como $(W'_6, \mathcal{R}'_6, V'_6)$, con $\{w_1^{IV}, w_3, w''_2, w''_1, w'_1, w'''_2, w'_3\} \in W'_6$; $\{(w''_1,\, w_1^{IV}),\, (w''_1,\, w'_1)\} \in R'_{6_{\bar{1}}}$, $\{(w_1^{IV},\, w''_1),\, (w'_1,\, w''_1)\} \in R'_{6_1}$, $\{(w_1^{IV}, w_3),\, (w''_2, w_1^{IV}),\, (w'_1, w'_3),\, (w'''_2, w'_1)\} \in R'_{6_2}$, $\{(w_3,\, w_1^{IV}),\, (w_1^{IV},\, w''_2),\, (w'_1, w''_2),\, (w'_3, w'_1)\} \in R'_{6_{\bar{2}}}$; $V'_6(q) = \{w_1^{IV},\, w''_1,\, w'_1\}$, $V'_6(t) = \{w_1^{IV},\, w''_1,\, w'_1\}$, $V'_6(s) = \{w_3,\, w'_3\}$ y $V'_6(p) = \{w''_2,\, w'''_2\}$. La estructura se presenta en la Figura 2.7.

f) Se establece $\mathfrak{M}'_1 = (W'_1, \mathcal{R}'_1, V'_1)$ indicado en el paso 1, la cual es la estructura final que retorna la función con el primer llamado, en donde en el nodo raíz w_1 se cumple ϕ_2, tal que $\{w_1^{IV}, w_3, w''_2, w''_1, w_1, w_2, w'''_2, w'_3\} \in W'_1$; $\{(w''_1,\, w_1^{IV}),\, (w''_1,\, w_1)\} \in R'_{6_{\bar{1}}}$, $\{(w_1^{IV},\, w''_1),\, (w_1,\, w''_1)\} \in R'_{6_1}$, $\{(w_1^{IV}, w_3),\, (w''_2, w_1^{IV}),\, (w_2,\, w_1),\, (w_1, w'_3),\, (w'''_2, w_1)\} \in R'_{6_2}$, $\{(w_3, w_1^{IV}),\, (w_1^{IV}, w''_2),\, (w_1,\, w_2),\, (w_1, w'''_2),\, (w'_3, w_1)\} \in R'_{6_{\bar{2}}}$; $V'_6(q) = \{w_1^{IV},\, w''_1,\, w_1\}, V'_6(t) = \{w_1^{IV},\, w''_1,\, w_1\}, V'_6(s) = \{w_3,\, w'_3\}$ y $V'_6(p) = \{w''_2,\, w_2,\, w'''_2\}$. La estructura final que satisface ϕ_2 se muestra en la Figura 2.8.

En la Figura 2.8 se observa la estructura con forma de árbol resultante $\mathfrak{M}'_1$. Esta se construye a partir de la estructura $\mathfrak{M}_2$ y la fórmula ϕ_2, presentes en la Figura 2.2. Ambas estructuras satisfacen ϕ_2 en el mundo w_1.

Capítulo 3

Algoritmo de satisfacción

La propiedad de modelo de árbol es la base para la creación de este algoritmo, debido a que determinar la satisfacción de una fórmula es posible mediante la búsqueda y, de ser el caso, la construcción de una estructura de Kripke con forma de árbol que la satisfaga en su nodo raíz. A nivel del algoritmo se emplea una estructura sintáctica de árbol que es definida en este capítulo en conjunto con algunos conceptos clave. Posteriormente, se detalla el algoritmo así como la prueba de corrección.

Definición 6. (Conjunto *Lean*). Dada una fórmula de lógica multimodal K con modalidades inversas ϕ en forma normal negativa, su conjunto *lean* (también expresado como $Lean(\phi)$) se define como la unión de:

- Las subfórmulas modales de ϕ, las subfórmulas de la forma $\neg p$ de ϕ y las variables proposicionales de ϕ que no estén negadas.
- Fórmulas $\langle\!\langle m \rangle\!\rangle \top$ y $[\![m]\!] \top$ por cada fórmula modal de modalidad m en ϕ.
- Una variable proposicional comodín E que no pertenezca a las subfórmulas de ϕ.

La variable proposicional comodín E ayuda a identificar posteriormente la ausencia de variables proposicionales y negación de variables proposicionales.

Ejemplo 3.1. Considere la fórmula $\phi_e = [\![1]\!](\neg p \wedge q) \wedge \langle\!\langle 1 \rangle\!\rangle(s \wedge [\![\overline{2}]\!](\neg q \vee \neg r))$, el conjunto $Lean(\phi_e)$ se compone de la siguiente manera:

$$\{[\![1]\!](\neg p \wedge q),\ \langle\!\langle 1 \rangle\!\rangle(s \wedge [\![\overline{2}]\!](\neg q \vee \neg r)),\ \langle\!\langle 1 \rangle\!\rangle \top,\ [\![1]\!] \top,\ \neg p,\ q,\ s,\ [\![\overline{2}]\!](\neg q \vee \neg r),\ \langle\!\langle 2 \rangle\!\rangle \top,\ [\![2]\!] \top,\ \neg q, \neg r,\ E\}$$

La siguiente definición especifica la estructura con la que trabajará el algoritmo, la cual se denomina como "estructura sintáctica de árbol". Ya que a diferencia de la estructura de Kripke, esta no es una tripleta, sino que es un árbol en el sentido estricto de la palabra, que contiene nodos, los cuales pueden contener fórmulas.

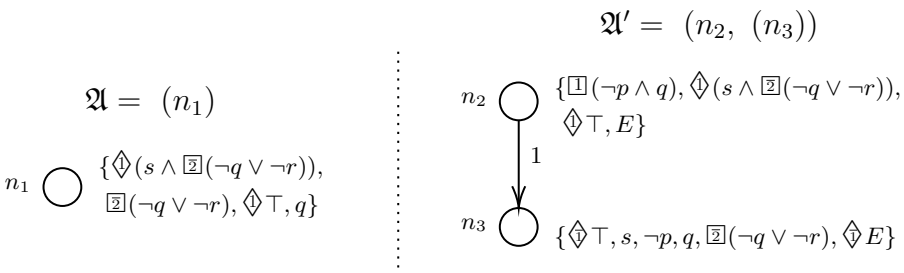


Figura 3.1: Ejemplos de estructuras sintácticas de árbol considerando el conjunto $Lean(\phi_e)$.

Definición 7. (Estructura sintáctica de árbol). Se define como una tupla $(raíz, \mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_n)$, tal que $raíz$ es el nodo raíz y $\mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_n$ son estructuras sintácticas de árbol, dadas las siguientes restricciones:

- Los nodos raíces de $\mathfrak{A}_1$, $\mathfrak{A}_2$, ..., $\mathfrak{A}_n$ son nodos hijos del nodo $raíz$.
- Una estructura sintáctica de árbol siempre debe tener un nodo raíz, y puede o no tener hijos, es decir, se pueden omitir las estructuras sintácticas $\mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_n$.
- Todos los nodos tienen un único nodo padre, a excepción del nodo raíz, que no posee ninguno.
- Todos los nodos hijos deben contener al menos una fórmula modal de la forma $\langle m\rangle\top$ cuando en el nodo padre está presente alguna fórmula $\langle m\rangle\top$, las cuales indican con que modalidad se relacionan los nodos.
- Todos los nodos h de una estructura sintáctica de árbol, sin considerar el nodo raíz, deben cumplir con lo siguiente:
  - En el nodo h debe estar incluida una fórmula de la forma $\varphi = \langle m\rangle(p_1 \wedge \ldots \wedge p_n)$, de manera que m es la modalidad de la fórmula $\langle m\rangle\top$ contenida en h y $p_1 \wedge \ldots \wedge p_n$ es la conjunción de todas las variables proposicionales presentes en el nodo padre de h.
  - En el nodo h deben estar presentes una o más fórmulas $\varphi_2 = \langle m\rangle\langle n\rangle \ldots \langle k\rangle(p_1 \wedge \ldots \wedge p_i)$ por cada fórmula modal de la forma $\varphi = \langle n\rangle \ldots \langle k\rangle(p_1 \wedge \ldots \wedge p_i)$ en el nodo padre de h, tal que m es la modalidad de la fórmula $\langle m\rangle\top$ contenida en h.

Ejemplo 3.2. En la Figura 3.1 se presentan dos estructuras sintácticas de árbol construidas acorde con la Definición 7, dada la fórmula $\phi_e = [1](\neg p \wedge q) \wedge \langle 1\rangle(s \wedge [2](\neg q \vee \neg r))$ tomada del ejemplo anterior. En el nodo n_2 de la estructura $\mathfrak{A}'$, la fórmula $\langle 1\rangle\top$ indica que debe existir al menos un nodo hijo que se relacione mediante la modalidad 1. Se puede observar como en el nodo n_3 existe la fórmula $\langle 1\rangle\top$, esto nos indica que la relación entre n_2 y n_3 se realiza con la modalidad 1.

n_1 $\{\langle\!\langle 1 \rangle\!\rangle(s \wedge [\![2]\!](\neg q \vee r)),\ \langle\!\langle 1 \rangle\!\rangle \top, E\}$ $\quad$ n_2 $\{\langle\!\langle 1 \rangle\!\rangle \top,\ \neg p\}$ $\quad$ n_3 $\{\langle\!\langle 1 \rangle\!\rangle \top,\ [\![2]\!](\neg q \vee \neg r),\ s,\ \langle\!\langle 1 \rangle\!\rangle E\}$

Figura 3.2: Ejemplos de nodos que se pueden generar dada la fórmula ϕ_e.

Definición 8. (Nodos). Un nodo de una estructura sintáctica de árbol tiene asociado un conjunto de fórmulas, de manera que se sostienen las siguientes condiciones:

- Los elementos del conjunto de fórmulas pertenecen a un subconjunto del $Lean(\phi)$. Adicionalmente, como excepción, los elementos también pueden ser fórmulas de la forma:
$$\langle\!\langle m \rangle\!\rangle \ldots \langle\!\langle n \rangle\!\rangle \Big(\bigwedge \mathcal{F} \Big)$$
tal que $\mathcal{F}$ corresponde al subconjunto de variables proposicionales de $\mathcal{VP}(\phi)$, además de la unión del subconjunto de las variables proposicionales de $\mathcal{VP}(\phi)$, pero aplicando el operador de negación a cada una.
- Debe existir al menos una variable proposicional o la negación de una variable proposicional.
- Si existe una fórmula modal $\langle\!\langle m \rangle\!\rangle \psi$ en el conjunto de fórmulas del nodo, entonces debe existir la fórmula $\langle\!\langle m \rangle\!\rangle \top$ en este mismo.
- Si existe una variable proposicional p, entonces no debe existir $\neg p$ y viceversa.

Ejemplo 3.3. Considere la fórmula del ejemplo anterior $\phi_e = [\![1]\!](\neg p \wedge q) \wedge \langle\!\langle 1 \rangle\!\rangle(s \wedge [\![2]\!](\neg q \vee \neg r))$. En la Figura 3.2 se muestran nodos válidos dadas las limitaciones impuestas por la Definición 8.

La próxima definición se requiere debido a que el algoritmo manipula estructuras sintácticas de árbol y no estructuras de Kripke. Por esta razón, se debe definir cuando una fórmula se cumple las nuevas estructuras que fueron especificadas en la Definición 7.

Definición 9. (Relación de consecuencia lógica $\Vdash$). Dada una fórmula ϕ y un nodo n que pertenece a una estructura sintáctica de árbol, la relación de consecuencia lógica se define como:

$$\frac{n \Vdash \phi_1,\ n \Vdash \phi_2}{n \Vdash \phi_1 \wedge \phi_2} \qquad \frac{n \Vdash \phi_1}{n \Vdash \phi_1 \vee \phi_2} \qquad \frac{n \Vdash \phi_2}{n \Vdash \phi_1 \vee \phi_2}$$

$$\frac{\phi \in n}{n \Vdash \phi} \qquad \frac{}{n \Vdash \top}$$

Cabe resaltar que las fórmulas modales, así como variables proposicionales y variables proposicionales negadas están consideradas en el primer caso de la segunda fila.

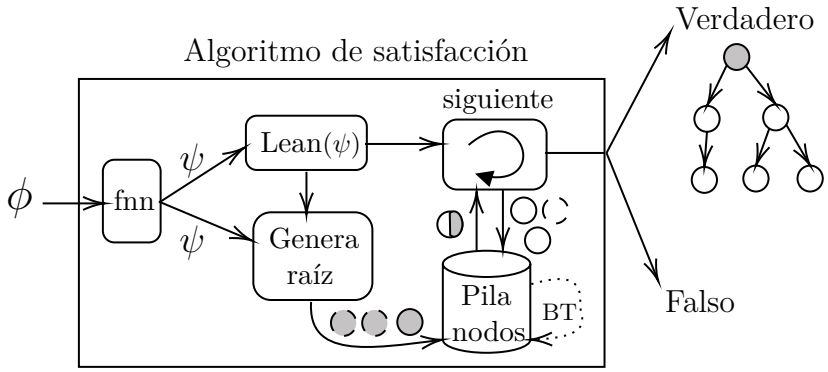


Figura 3.3: Diagrama que muestra el funcionamiento principal del algoritmo.

Ejemplo 3.4. Considere las estructuras sintácticas de árbol mostradas en la Figura 3.1 y la fórmula $\phi_e = \boxed{1}(\neg p \wedge q) \wedge \diamondsuit_1(s \wedge \boxed{2}(\neg q \vee \neg r))$ tomada de ejemplos anteriores. Se puede verificar que en la estructura sintáctica de árbol de la izquierda no se cumple $n_1 \Vdash \phi_e$. Esto es debido a que la regla de la conjunción indica que debe estar presente en el nodo n_1 tanto $\boxed{1}(\neg p \wedge q)$ como $\diamondsuit_1(s \wedge \boxed{2}(\neg q \vee \neg r))$, lo cual no ocurre.

Por otro lado, sí se verifica $n_2 \Vdash \phi_e$ en la estructura sintáctica de árbol de la derecha de la misma figura, se observa que en n_2 están presentes $\boxed{1}(\neg p \wedge q)$ y $\diamondsuit_1(s \wedge \boxed{2}(\neg q \vee \neg r))$, además como hay una fórmula modal de necesidad existe $\diamondsuit_1 \top$ y como no se cumple ninguna variable proposicional está la variable comodín E para cumplir con la Definición 8, que indica que debe existir al menos una variable proposicional o la negación de una variable proposicional.

A continuación, se presenta el algoritmo en un nivel general con ayuda de la Figura 3.3. Como primer paso se requiere que la fórmula de entrada ϕ sea transformada a su forma normal negativa ψ. Posteriormente, se debe generar el conjunto *lean* dada ψ. El bloque que genera los posibles nodos raíces recibe como entrada el conjunto de fórmulas generadas del *lean* y la fórmula ψ, con base en esos dos elementos genera los posibles nodos raíces a probar para construir la estructura sintáctica de árbol.

La pila almacena los nodos generados, esta estructura permite tener un registro de los nodos pendientes de incluir o probar en la estructura sintáctica de árbol, además la información almacenada ayuda a realizar una vuelta atrás (*backtracking*[1] o BT) cuando haya nodos que contengan contradicciones. Mientras la pila de nodos no esté vacía, un nodo es enviado como entrada a la función `siguiente`. Esta función se encarga de generar los nodos hijos y todas sus posibilidades de manera consistente, agrega los nodos necesarios al árbol y los demás se envían a la pila. Además, se encarga de realizar la vuelta atrás

[1]Es una técnica que explora todas las posibles soluciones de un problema, de manera que en cada paso se determina si el camino que se está explorando tiene potencial para terminar en una solución correcta, en caso de no ser así se retrocede y se prueba otro camino.

en la pila cuando sea requerido o inclusive volver a generar los nodos raíces con nueva información.

Si la pila está vacía, entonces el algoritmo retorna un valor de verdad y de ser verdadero, una estructura sintáctica de árbol. A continuación se presenta la especificación del algoritmo de satisfacción de manera precisa.

En términos generales el algoritmo de satisfacción se compone de 4 funciones clave: `generarRaiz`, `siguiente`, `valNodosRep` y `vueltaAtrás`. La primera de ellas se ejecuta por primera vez antes de ingresar al ciclo principal, genera todos los posibles nodos raíces iniciales de las estructuras sintácticas de árbol a comprobar.

Algoritmo 1 Satisfacción.

Entrada: Una fórmula de la lógica multimodal K con inversas $\phi_{entrada}$.
Salida: Retorna verdadero y una estructura sintáctica de árbol, si y solo si, $\phi_{entrada}$ se satisface bajo alguna estructura de Kripke de tipo árbol en el nodo raíz, en caso contrario, devuelve falso.

```
1: pilaNodos ← [ ], backtrackNodos ← [ ]
2: φ ← aFNN(φ_entrada)
3: lean ← Lean(φ)
4: raizNodos ← generarRaiz(φ, lean)

5: if raizNodos = [ ] then
6:     return false
7: end if
8: arbol ← raizNodos[0]
9: pilaNodos ← append(pilaNodos, reverse(raizNodos))

10: while pilaNodos ≠ [ ] do
11:     nodoActual ← pop(pilaNodos)
12:     if nodoActual_esAlt = false && nodoActual_esSB = false && nodoActual_desact
          = false then
13:         if nodoActual_esSD = true && |pilaNodos| > 0 then
14:             nodoSig ← head(pilaNodos)
15:             if (nodoSig_esAlt = true || nodoSig_esSB = true) && nodoSig_esSD = false
    then
16:                 backtrackNodos ← append(backtrackNodos, nodoSig)
17:             end if
18:         end if
```

```
19:         if nodoActual_padre = None then
20:             arbol ← nodoActual
21:         end if
22:         insat, nodosHijos ← siguiente(nodoActual, lean)

23:         if insat = false then
24:             hayNodosRep, nodosHijos ← valNodosRep(arbol, nodosHijos)
25:         end if
26:         if (nodosHijos = [ ] && hayNodosRep = true) || (insat = true && nodosHijos
            = [ ] && (pilaNodos = [ ] || backtrackNodos = [ ])) then
27:             return false
28:         end if

29:         if insat = true then
30:             hayAlt, arbol ← vueltaAtrás(nodoActual, pilaNodos, backtrackNodos,
                arbol)
31:             if hayAlt = false then
32:                 return false
33:             end if
34:         end if
35:         pilaNodos ← append(pilaNodos, nodosHijos)
36:     else
37:         if nodoActual_esSB = true && |backtrackNodos| > 0 &&
            head(backtrackNodos) = nodoActual then
38:             pop(backtrackNodos)
39:         end if
40:     end if
41: end while
42: return true, arbol
```

La función `siguiente` es la más importante, debido a que se encarga de generar nuevos nodos tomando un nodo de referencia, que en su primer llamado siempre es un nodo raíz y realiza la unión a la estructura sintáctica de árbol para la primer ronda de posibles nodos hijos. La tercera función verifica que los nodos nuevos no se encuentren repetidos en el árbol que se está construyendo, de lo contrario el proceso termina. Finalmente `vueltaAtrás` realiza una vuelta atrás cuando sea necesario y siempre que sea posible, de lo contrario, la fórmula no se satisface.

Al final de cada definición se presentará un ejemplo con una entrada para la función descrita y la salida que produce.

Definición 10. La función `generarRaiz(`ϕ`, lean)` posee dos parámetros de entrada: una fórmula en forma normal negativa ϕ y el conjunto lean generado al inicio del algo-

$$\phi_3 = \langle 1 \rangle (p \wedge q) \vee [2](r \vee s) \vee \neg q$$

$$Lean(\phi_3) = \{\langle 1 \rangle (p \wedge q),\ p,\ q,\ [2](r \vee s),\ r,\ s,\ \neg q,\ \langle 1 \rangle \top,\ \langle 1 \rangle \top,\ \langle 2 \rangle \top,\ \langle 2 \rangle \top,\ E\}$$

$$\texttt{generarRaiz}(\phi_3,\ Lean(\phi_3)) = n_1 \bigcirc\ [\{\langle 1 \rangle (p \wedge q),\ E,\ \langle 1 \rangle \top\},$$
$$n_2 \cdots\ \{[2](r \vee s),\ E,\ \langle 2 \rangle \top\},$$
$$n_3 \cdots\ \{\neg q,\ E\}]$$

Figura 3.4: Salida de `generarRaiz` dada ϕ_3 y $Lean(\phi_3)$ como entrada.

ritmo. Esta función construye los posibles nodos raíz que se emplearán para iniciar la construcción de la estructura sintáctica de árbol. El proceso preciso que realiza se especifica a continuación:

- Se obtienen las fórmulas que deberán satisfacerse en todos los posibles nodos raíces, derivado de la relación de consecuencia lógica $\Vdash \phi$ de la Definición 9, y se almacena cada conjunto de fórmulas en un conjunto llamado `formulasNodo`, con excepción de aquellos conjuntos que contengan contradicciones con respecto a sus variables proposicionales.
- Si el conjunto $\texttt{formulasNodo} = \emptyset$, entonces:
  - La función devuelve [ ].
- Por cada fórmula modal $\psi = \langle m \rangle \varphi$ perteneciente a cada conjunto contenido en `formulasNodo`:
  - Se agrega $\langle m \rangle \top$ a $\texttt{formulasNodo}_i$.
- Se crea una lista vacía llamada `raizNodos`.
- Por cada conjunto en `formulasNodo`, se realiza lo siguiente:
  - Si no existen variables proposicionales en $\texttt{formulasNodo}_i$, entonces agrega la variable proposicional comodín E a ese mismo conjunto.
  - Si el conjunto de fórmulas $\texttt{formulasNodo}_i$ cumple con todas las restricciones para poder construir un nodo consistente, entonces se crea un nodo $\texttt{n}_i$, que posee la información de ese conjunto de fórmulas, y se almacena en la lista `raizNodos`.
- Al primer nodo `nP` de la lista de nodos `raizNodos` se le asigna la bandera $\texttt{nP}_{esAlt} \leftarrow false$ y a los demás nodos `nA` de la lista de nodos `raizNodos` se les establece $\texttt{nA}_{esAlt} \leftarrow true$.
- La función devuelve el conjunto de nodos creados: `raizNodos`.

Ejemplo 3.5. Considere la fórmula ϕ_3 de la Figura 3.4 y el conjunto $Lean(\phi_3)$ como entrada para la función `generarRaiz`. Se puede observar que a la salida la función produce 3 posibles nodos raíz. Estos nodos se generan acorde a lo estipulado en las Definiciones 8 y 9 (Nodos y Relación de consecuencia lógica). El operador de disyunción es el causante de que se genere más de un nodo.

Definición 11. La función `siguiente` recibe como parámetros el `nodoActual` y el conjunto *lean*. La expresión $\texttt{nodoActual}_{formulas}$ corresponde a las fórmulas que están contenidas en el `nodoActual`. Al terminar, de ser posible crear los nodos hijos del nodo actual, éstos serán consistentes y la variable `insat` será falsa, la función retorna `insat` y una lista llamada `nodosHijos`. En caso contrario, si el `nodoActual` posee alguna contradicción, retorna `insat` con un valor verdadero y una lista vacía. Los detalles se presentan a continuación:

Si $\texttt{nodoActual}_{formulas} = \emptyset$, entonces:

- Retorna verdadero (`insat`) y una lista vacía (`nodosHijos`).

Si $\texttt{nodoActual}_{nivel} > nivelActual$:

- Si el $nivelActual = 0$, entonces llama a la función `genAncesFórmulas`($lean$)
- En caso contrario, llama a la función `genAncesFórmulas`($lean$).

Si `nodoActual` es un nodo especial de un caso de modalidad diamante ($\texttt{nodoActual}_{esSD} = true$), entonces:

- Si existe algún nodo `nR` que pertenece a la lista de nodos $\texttt{nodoActual}_{RN}$, la cual es una lista que contiene todos los nodos asociados a la misma rama y el mismo nivel que el `nodoActual`, y existe algún nodo `nC`, contenido en la lista de nodos hijos del padre de `nodoActual`, tal que $\texttt{nC} \neq \texttt{nR}$, $\texttt{nC}_{esAlt} = false$, `nC` se encuentra a la izquierda del `nodoActual` en la lista de nodos del padre de `nodoActual` y el conjunto de fórmulas de `nR` es un subconjunto de $\texttt{nC}_{formulas}$, entonces:
  - El `nodoActual` se remueve de la lista de nodos hijos de su padre.
  - Todos los nodos `nR` de la lista $\texttt{nodoActual}_{RN}$ se deshabilitan estableciendo la bandera ($\texttt{nR}_{desact} \leftarrow true$), además si algún nodo `nR` ya estaba como nodo hijo del nodo padre de `nodoActual`, lo remueve de esa lista de nodos hijos.
- En caso contrario:
  - Se modifica la bandera del primer nodo `nR0` de la lista $\texttt{nodoActual}_{RN}$ de la siguiente manera: $\texttt{nR0}_{esAlt} \leftarrow false$.
  - Se remplaza el `nodoActual` de la lista de hijos de su nodo padre por el nodo `nR0`.
- Se establece $\texttt{nodoActual}_{desact} \leftarrow true$.

- Retorna `insat` con un valor de falso y una lista vacía de `nodosHijos`.

Se establece la bandera `reqRevCBIS` con un valor booleano falso.

Si `nodoActual` no es un nodo raíz y la longitud de la lista de nodos $\texttt{nodoActual}_{refNodes}$ es mayor a cero (dicha lista contiene una lista de conjuntos de fórmulas que originaron un cambio en el `nodoActual` debido a un caso de caja con modalidad inversa), entonces:

- Se establece la bandera `reqRevCBIS` a verdadero.

Si $\texttt{nodoActual}_{formulas} \neq \emptyset$, entonces:

- Se crean los conjuntos vacíos: `formulasCaja` y `formulasDiamante`, así como una lista vacía `nodosHijos`.
- Por cada fórmula modal $\psi = \diamondsuit_{m} \varphi$ *o* $\boxed{m} \varphi$ contenida en el conjunto $\texttt{nodoActual}_{formulas}$, excepto aquellas fórmulas φ que no sean subfórmulas de la fórmula de entrada ϕ, se realiza lo siguiente:
  - Se crea un conjunto vacío auxiliar: `formulasNodo`.
  - Se obtienen todos los posibles conjuntos de fórmulas que deberán estar contenidos en cada uno de los futuros nodos hijos acorde a la Definición 9 ($\Vdash \varphi$) y se almacenan en el conjunto `formulasNodo`. Además, si la fórmula posee la forma $\psi = \diamondsuit_{m} \varphi$, se remueven aquellos conjuntos de `formulasNodo` que contengan contradicciones con respecto a sus variables proposicionales.
  - Si el conjunto $\texttt{formulasNodo} = \emptyset$, entonces:
    - Retorna verdadero (`insat`) y una lista vacía (`nodosHijos`).
  - Por cada conjunto de fórmulas en `formulasNodo` se realiza lo siguiente:
    - Agrega $\diamondsuit_{m} T$ a $\texttt{formulasNodo}_i$, dada la modalidad m de ψ.
    - Por cada fórmula modal $\psi_1 = \langle m_1 \rangle \varphi_1$ o $[m_1] \varphi_1$ en $\texttt{formulasNodo}_i$, se agrega la fórmula $\langle m_1 \rangle \top$ a $\texttt{formulasNodo}_i$.
    - Si no existen variables proposicionales en $\texttt{formulasNodo}_i$, entonces agrega E a $\texttt{formulasNodo}_i$.
  - Si $\psi = \boxed{m} \varphi$, entonces:
    - Si en el conjunto `formulasCaja` existe una tupla (mFC, `formulasFC`), tal que $m = mFC$, entonces:
      - $\texttt{formulasFC} \leftarrow \{\{F_1 \cup \ldots \cup F_k\} \mid (F_1, \ldots, F_k) \in (\texttt{formulasFC} \times \texttt{formulasNodo})\}$, donde k = $|(\texttt{formulasFC} \times \texttt{formulasNodo})|$.
      - Por cada conjunto contenido en `formulasFC` se realiza lo siguiente: Si hay variables proposicionales en $\texttt{formulasFC}_i$, entonces: $\texttt{formulasFC}_i \leftarrow \texttt{formulasFC}_i \backslash \{E\}$.
    - En caso contrario:

◇ Agrega la tupla (m, `formulasNodo`) al conjunto `formulasCaja`, tal que m es la modalidad en ψ.

- Si $\psi = \Diamond_m \varphi$, entonces:
  - Por cada conjunto de fórmulas en `formulasNodo` se realiza lo siguiente:
    - ◇ Agrega la fórmula $\Diamond_m \alpha$, tal que α tiene la forma $\beta_0 \wedge \beta_1 \wedge \ldots \beta_n$, las fórmulas $\beta_0, \beta_1, \ldots, \beta_n$ tienen la forma p o $\neg p$, de manera que α posee conjunciones de todas las variables proposicionales y negación de las variables proposicionales que están presentes en el conjunto $\texttt{nodoActual}_{formulas}$.
      Además, por cada fórmula modal $\Diamond_n$, ..., $\Diamond_{n_i}\gamma$, tal que γ tiene la forma de α, presente en el conjunto $\texttt{nodoActual}_{formulas}$, se procede a agregar al conjunto $\texttt{formulasNodo}_i$ la siguiente fórmula: $\Diamond_m \Diamond_n$, ..., $\Diamond_{n_i}\gamma$.
  - Por cada conjunto contenido en `formulasDiamante`, si $\texttt{formulasDiamante}_i$ contiene una fórmula modal de la forma $\langle\overline{mFD}\rangle\top$, de manera que $mFD = m$ entonces agrega el conjunto `formulasNodo` a $\texttt{formulasDiamante}_i$. En caso contrario, si en ningún conjunto $\texttt{formulasDiamante}_i$ hay una fórmula modal $\langle\overline{mFD}\rangle\top$, entonces agrega el conjunto `formulasNodo` a `formulasDiamante`.

- Considere cada conjunto contenido en `formulasDiamante`, que será nombrado $\texttt{formulasDiamante}_i$, y k como la referencia a un elemento dentro de este último:
  - Se calcula el conjunto potencia (`fsubsets`) de los conjuntos contenidos en $\texttt{formulasDiamante}_{i,k}$, si $k > (i+1)$ y $k < |\texttt{formulasDiamante}_i|$.
  - Se agrega $\texttt{formulasDiamante}_{i,j}$ a cada elemento del conjunto `fsubsets`, finalmente se agrega $\texttt{formulasDiamante}_{i,j}$ al conjunto `fsubsets`.
- Se crea una lista auxiliar vacía `formulasNodes`.
- Por cada elemento de `fsubsets` se realiza lo siguiente:
  - Por cada conjunto contenido en $\texttt{fsubsets}_i$, se almacena en la lista `formulasNodoith` la unión de todos los conjuntos de $\texttt{fsubsets}_{i,j}$. Después, se agrega el conjunto `formulasNodoith` a `formulasNodes`.
  - Por cada conjunto de fórmulas de `formulasNodes` se realiza lo siguiente: si $\texttt{formulasNodes}_i$ contiene alguna contradicción con respecto a las variables proposicionales contenidas, entonces se elimina de `formulasNodes`.
  - Se crea una lista vacía `nodosHS`.
  - Por cada conjunto de `formulasNodes`, se crea un nodo fnH con la información de $\texttt{formulasNodes}_j$, el primer nodo creado tendrá asignada $fnH_{esAlt} \leftarrow false$, los demás nodos tendrán $fnH_{esAlt} \leftarrow true$ y cada nodo se almacenará en la lista `nodosHS`

    - Se crea un nodo especial `nSD`, tal que $\texttt{nSD}_{esAlt} \leftarrow true$, $\texttt{nSD}_{esSD} \leftarrow true$ y $\texttt{nSD}_{refNodes} \leftarrow \texttt{nodosHS}$.
    - Se agrega la lista `nodosHS` a la lista `nodosHijos`.

- Por cada tupla ($mPos$, `formulasPos`) en `formulasCaja` se realiza lo siguiente:
    - Si la longitud de la lista $\texttt{nodoActual}_{refNodes}$ es mayor que cero, entonces:
        - Por cada nodo, que será nombrado `rNRama`, en $\texttt{nodoActual}_{refNodes}$ se realiza lo siguiente:
            - La primera fórmula en $\texttt{rNRama}_{formulas}$ tiene la forma $\langle \overline{n} \rangle \top$. Se extrae la modalidad n y se compara con la modalidad $mPos$, si ambas coinciden entonces se crea una copia del nodo `rNRama` que será llamada `copiaNR`, por último, se realiza $\texttt{copiaNR} \Vdash \texttt{formulasPos}$ según la Definición 9 y se remplaza el nodo `rNRama` por `copiaNR` en la lista $\texttt{nodoActual}_{refNodes}$.
    - Por cada lista de nodos `nHS` en `nodosHijos` se realiza lo siguiente:
        - Si $mH = mPos$, tal que mH es la modalidad con la que se relaciona algún nodo hijo en `nHS` con el `nodoActual`, es decir, si dicho nodo hijo posee una fórmula de la forma $\langle \overline{mH} \rangle \top$, se lleva a cabo lo siguiente:
            - Se crean las listas auxiliares `newNodesBox` ← [ ] y `refNodesSD` ← [ ].
            - Por cada conjunto de `formulasPos`, se efectúa lo siguiente:
            - * Se copia la lista `nHS` a una nueva lista llamada `cHS`, se eliminan los nodos `cN` de `cHS` que posean la bandera $\texttt{cN}_{esSD} = true$.
            - * Por cada nodo `cH` de la lista `cHS` se intenta realizar $\texttt{cH} \Vdash \texttt{formulasPos}_i$ y se eliminan aquellos nodos que posean contradicciones de la lista `cHS`, así como repetidos.
            - * Se concatenan los nodos de `cHS` a la lista `refNodesSD`.
            - * Si la lista de nodos `cHS` posee al menos un nodo, se agrega `cHS` a la lista `newNodesBox`.
            - Si dentro de la lista `nHs` existe un nodo especial `nSD`, que tiene la bandera $\texttt{nSD}_{esSD} = true$, entonces se establece $\texttt{nSD}_{RN} \leftarrow \texttt{refNodesSD}$.
            - Se agrega un nodo especial `nSB` que no contiene fórmulas y tiene la bandera $\texttt{nSB}_{esSB} \leftarrow false$ al final de la lista `newNodesBox`.
            - Si `nHS` es el primer elemento de la lista `nodosHijos`, se establece lo siguiente $\texttt{nSB}_{princialSigBoxNodos} \leftarrow \texttt{newNodesBox}$.
            - En caso contrario, asigna $\texttt{nSB}_{princialSigBoxNodos} \leftarrow [\,]$.
            - Si el tamaño de la lista $\texttt{newNodesBox} = 0$ o únicamente contiene nodos con la bandera $esSD = true$ o $esSB = true$, entonces:
            - * Retorna verdadero (`insat`) y una lista vacía (`nodosHijos`).
            - Se actualiza la lista de nodos `nHS` que está presente en `nodosHijos` como `nHS` ← `newNodesBox`.

- Se definen e inicializan `iteracionesMax` $\leftarrow 1$, `vNodoActual` $\leftarrow$ `nodoActual` y `vNodosHijos` $\leftarrow$ `nodosHijos`.
- Si la bandera `reqRevCBIS` posee un valor verdadero:
  - Se modifica el valor `iteracionesMax` $\leftarrow 2$.
  - Se crea una lista auxiliar llamada `auxNodosS` que contendrá el `nodoActual` y los nodos alternativos. Es decir, se obtiene el nodo `nHAux` $\leftarrow$ `nodoActual`$_{derecha}$ y mientras $nHAux$ no tenga un valor nulo se agrega `nHAux` a la lista $auxNodosS$ y se actualiza `nHAux` $\leftarrow$ `nodoActual`$_{derecha}$.
  - Se establece `vNodoActual` $\leftarrow$ `nodoActual`$_{padre}$ y `vNodosHijos` $\leftarrow$ [`auxNodosS`].
- Se crean las listas `todoFormulas` $\leftarrow$ [ ] y `todoRefNodes` $\leftarrow$ [ ].
- Mientras `iteracionesMax` > 0 se realiza lo siguiente:
  - Si `iteracionesMax` $= 0$ se actualiza la bandera `reqRevCBIS` $\leftarrow false$.
  - Se crea un conjunto auxiliar `removeNodes` $= \emptyset$.
  - Se crean las listas `ramaFormulas` $\leftarrow$ [ ] y `ramaRefNodes` $\leftarrow$ [ ].
  - Por cada lista de nodos `vLNH` en `vNodosHijos` y por cada nodo `vNH` en `vLNH`:
    - Se crean las bandera `nodoCumpleEnPadre` $\leftarrow false$ y `hayCasoEspecial` $\leftarrow false$.
    - Se verifica si todos los elementos del conjunto `vLNH` están presentes en algún conjunto `vNodoActual`$_{refNodes}$, de ser así, se establece la bandera `nodoCumpleEnPadre` a verdadero. Esta bandera permite descartar nodos los cuales tienen fórmulas pendientes por propagar al nodo padre, pero que ya se habían probado en el árbol.
    - Se crea una variable auxiliar `formulasCNCBIS` $= \emptyset$.
    - Se escanean las fórmulas del conjunto `vLNH`$_{formulas}$ en busca de fórmulas de la forma $\boxed{m}\varphi_b$, tal que m proviene de ψ (que corresponde a una fórmula modal definida anteriormente). Si se encuentra alguna de ese tipo, se habilita la bandera `hayCasoEspecial` $\leftarrow true$. se obtienen las fórmulas `formulasAuxCB` que deberán estar presentes en un futuro nodo $\Vdash \varphi_b$, dada la Definición 9. Posteriormente se realiza la operación `formulasCNCBIS` $\leftarrow \{\{F_1 \cup \ldots \cup F_k\} \mid (F_1, \ldots, F_k) \in (\texttt{formulasCNCBIS} \times \texttt{formulasAuxCB})\}$, donde k $= |(\texttt{formulasCNCBIS} \times \texttt{formulasAuxCB})|$.
    - Se almacena el conjunto `formulasAuxCB` en `formulasCNCBIS`.
    - Se verifica si $|\texttt{vNodoActual}_{refNodes}| > 0$, `nodoCumpleEnPadre` $= false$ y `hayCasoEspecial` $= true$, entonces se agrega el nodo `vLNH` al conjunto `removeNodes`. De no cumplirse lo anterior, se verifica si `hayCasoEspecial` $= true$ y se realiza lo siguiente:
      - Se crea la bandera `unoEncontrado` $\leftarrow false$.

◇ Se verifica cada conjunto de `formulasCNCBIS`, si cada una de las fórmulas en algún conjunto $\texttt{formulasCNCBIS}_i$ están presentes en el conjunto $\texttt{vNodoActual}_{formulas}$, entonces se establece la bandera `unoEncontrado` $\leftarrow$ $true$.

◇ Si `unoEncontrado` $= false$, se verifica si el conjunto `formulasCNCBIS` está presente en la lista `ramaFormulas`, de ser así, se almacena el nodo `vNH` en la lista $\texttt{ramaRefNodes}_j$, en la misma posición j-ésima en la que se encuentra `formulasCNCBIS` en la lista `ramaFormulas`. En caso contrario, se agrega el conjunto `formulasCNCBIS` a la lista `ramaFormulas` y se incluye $[vNH]$ en la lista `ramaRefNodes`.

○ Si `hayCasoEspecial` $= false$

◇ Se verifica si el conjunto $\emptyset$ está presente en la lista `ramaFormulas`, de ser así, se almacena el nodo `vNH` en la lista $\texttt{ramaRefNodes}_j$, en la misma posición j-ésima en la que se encuentra $\emptyset$ en la lista `ramaFormulas`. En caso contrario, se agrega el conjunto $\emptyset$ a la lista `ramaFormulas` y se incluye $[vNH]$ en la lista `ramaRefNodes`.

○ Al finalizar la iteración con `vNH`, se agrega la lista `ramaFormulas` a la lista `todoFormulas` y `ramaRefNodes` a `todoRefNodes`

- Se crea una lista vacía `nodosPadreOri` $\leftarrow$ [ ] y una variable auxiliar `auxNPO` $\leftarrow$ `vNodoActual`.
- Comienza un proceso iterativo mientras `auxNPO` sea distinto de un valor nulo, entonces:
  - Se agrega `auxNPO` a la lista `nodosPadreOri`.
  - Se altera el valor de la variable `auxNPO` $\leftarrow$ $\texttt{auxNPO}_{derecha}$. El nodo que se obtiene de $\texttt{auxNPO}_{derecha}$ corresponde a una alternativa de `auxNPO`, es decir un posible nodo para esa misma rama en el mismo nivel.
- Se define un conjunto `auxTLF` $= \emptyset$ y una lista `finalRefNodes` $\leftarrow$ [ ].
- Por cada lista contenida en `todoFormulas` se efectúa lo siguiente:
  - Se define un conjunto `RLF` $= \emptyset$ y una lista `RN` $\leftarrow$ [ ].
  - Por cada conjunto contenido en la lista $\texttt{todoFormulas}_i$ se realiza lo siguiente:
    - ◇ Se agrega cada elemento de $\texttt{todoFormulas}_{i,j}$ al conjunto `RLF`.
    - ◇ Se incluye cada lista contenida en `todoRefNodes` a la lista `RN`.
  - Se realiza la siguiente operación `auxTLF` $\leftarrow \{\{F_1 \cup \ldots \cup F_k\} \mid (F_1, \ldots, F_k) \in (\texttt{auxTLF} \times \texttt{RLF})\}$, donde k $= |(\texttt{auxTLF} \times \texttt{RLF})|$. Esta misma operación de conjuntos se realiza de manera análoga con listas, en el caso de `finalRefNodes` y `RN`.
- Si el tamaño del conjunto `auxTLF` $= 0$, indica que en el caso de los nodos hijos que poseían operadores caja con inversa, y que originaba un cambio en el nodo padre, las subfórmulas de esas fórmulas modales ya se cumplen en el nod padre. Se procede a realizar lo siguiente:

- Si `reqRevCBIS` $= true$, entonces se establece: `vNodoActual` $\leftarrow$ `nodoActual` y `vNodosHijos` $\leftarrow$ `nodosHijos`.
- Si el tamaño del conjunto `removeNodes` es mayor que cero, entonces se realizará una búsqueda sobre los nodos contenidos en `vNodosHijos` y si coincide con alguno del conjunto `removeNodes` entonces se remueve de la lista `vNodosHijos`. El proceso de eliminación se debe hacer de manera que si se terminan los nodos de una rama en particular, entonces la función regresa verdadero (`insat`) y una lista vacía (`nodosHijos`) para proceder a realizar una vuelta atrás. Además, si se modifica una rama que contiene un nodo con la bandera SB en verdadero, se debe actualizar su lista `princialSigBoxNodos` conteniendo los nodos de la rama sin considerar aquellos que se eliminaron.

- En caso contrario, se procede con lo siguiente:
  - Se crea una lista de nodos llamada `nodosFrescos` $= [\ ]$.
  - Por cada conjunto contenido en `auxTLF` se realiza lo siguiente:
    - Por cada fórmula modal de la forma $\Diamond_m \varphi_a$ contenida en `auxTLF`$_i$, se agrega $\Diamond_m \top$ a `auxTLF`$_i$.
    - Se crea una copia de la lista `nodosPadreOri` llamada `copiaNodosPO`.
    - Cada nodo `cNPO`, contenido en la lista `copiaNodosPO`, se actualiza de la siguiente manera `cNPO`$_{refNodes} \leftarrow finalRefNodes_i$.
    - Si el tamaño del conjunto `auxTLF`$_i$ es mayor a uno, se agregan a cada uno de los conjuntos de fórmulas de `copiaNodosPO`$_{formulas}$ las fórmulas de `auxTLF`$_i$. Si se producen nodos repetidos se remueven de la lista `copiaNodosPO`.
    - Se agregan todos los nodos de `copiaNodosPO` a la lista `nodosFrescos`.
  - El primer nodo `pNF` contenido en `nodosFrescos` se le modifica la bandera `pNF`$_{esAlt} = false$ y al resto de nodos se establece la bandera `esAlt` $= true$.
  - Por cada nodo nF en la lista `nodosFrescos` se efectúa lo siguiente:
    - Se establece el nodo padre `nF`$_{padre} \leftarrow$ `vNodoActual`$_{padre}$.
    - Se actualiza la referencia derecha del nodo `nF`$_{derecha} \leftarrow$ `nFSD`, tal que `nFSD` se encuentra en la posición siguiente con respecto de `nF` en la lista `nodosFrescos`.
    - Se modifica la referencia izquierda del nodo `nF`$_{izquierda} \leftarrow$ `nFSI`, tal que `nFSI` es el nodo en la posición anterior con respecto de `nF` en `nodosFrescos`.
  - Si el tamaño de la lista `nodosFrescos` es mayor que cero, el algoritmo retorna verdadero (`insat`) y `nodosFrescos` (`nodosHijos`). Esto con el objetivo de realizar una vuelta atrás y reemplazar los nodos padre.

- Por cada lista de `nodosHijos`, se establece el primer nodo de dicha lista como hijo del `nodoActual`.

$$\phi_4 = \langle 1 \rangle (p \vee [2]r) \wedge \langle 2 \rangle [2] t \wedge \langle 1 \rangle (\neg p \vee \langle 2 \rangle q)$$

$$Lean(\phi_4) = \{\langle 1 \rangle (p \vee [2]r),\ p,\ [2]r,\ r,\ \langle 2 \rangle [2] t,\ [2]t,\ t,\ \langle 1 \rangle (\neg p \vee \langle 2 \rangle q),$$
$$\neg p,\ \langle 2 \rangle q,\ q,\ \langle 1 \rangle \top,\ \langle 1 \rangle \top,\ \langle 2 \rangle \top,\ \langle 2 \rangle \top,\ E\}$$

n_1 ◯ $\{\langle 1 \rangle (p \vee [2]r),\ \langle 2 \rangle [2] t,\ \langle 1 \rangle (\neg p \vee \langle 2 \rangle q),\ \langle 1 \rangle \top,\ \langle 2 \rangle \top,\ E\}$

$$\texttt{siguiente}(n_1,\ Lean(\phi_4)) = [\{\langle 1 \rangle \top,\ p,\ \langle 2 \rangle q,\ \langle 2 \rangle \top, \langle 1 \rangle E\}^{n_2},\ \{\langle 1 \rangle \top,\ [2]r,\ p,\ \langle 2 \rangle \top,\ \langle 1 \rangle E\}^{n_3},$$

Nodo que no se genera debido a contradicción:

~~$\{\langle 1 \rangle \top,\ p,\ \neg p,\ \langle 1 \rangle E\}$~~

$$\{\langle 1 \rangle \top,\ [2]r,\ \langle 2 \rangle q,\ \langle 2 \rangle \top,\ \langle 1 \rangle E,\ E\}^{n_4},\ \emptyset^{n_5},\ \{\langle 1 \rangle \top,\ \neg p,\ \langle 1 \rangle E\}^{n_6},$$
$$\{\langle 1 \rangle \top,\ \langle 2 \rangle q,\ \langle 1 \rangle E,\ E\}^{n_7}, \{\langle 2 \rangle \top,\ [2]t,\ \langle 2 \rangle E,\ E\}^{n_8}]$$

$$insat\ =\ false$$

Estructura sintáctica de árbol después de ejecutar `siguiente`:

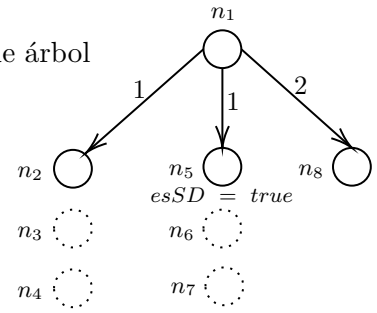


Figura 3.5: Ejemplo de la salida de la función `siguiente` ante un nodo n_1 proveniente de ϕ_4.

- Por cada lista de nodos $\{nH_1,\ \ldots,\ nH_m\}$ contenida en `nodosHijos` se efectúa lo siguiente: cada nodo nH_j posee una referencia $\texttt{nH}^j_{rightN} \leftarrow \texttt{nH}^{j-1}$, con $j > 1$, $j <= |\texttt{nodosHijos}_i|$ y $m = |\texttt{nodosHijos}|$. Además, el primer nodo $i = 1$ posee la bandera $\texttt{nH}^1_{principal} \leftarrow true$ y los demás nodos $i \neq 1$ tienen la bandera $\texttt{nH}^1_{principal} \leftarrow false$.
- Toma cada lista de `nodosHijos`, las concatena en una única lista manteniendo el orden. La función retorna falso y dicha lista de nodos hijos.

Ejemplo 3.6. En la Figura 3.5 se observa a la función `siguiente` recibir un nodo y el conjunto *lean* de una fórmula de entrada. Esta función genera el siguiente nivel de nodos hijos tomando en cuenta el nodo recibido.

Cada fórmula modal diamante produce una nueva rama, el nodo n_1 posee tres: $\langle 1 \rangle (p \vee [2]r)$, $\langle 2 \rangle [2] t$ y $\langle 1 \rangle (\neg p \vee \langle 2 \rangle q)$. La función comienza a crear las ramas dado el orden de la fórmula recibida, no obstante las ramas de misma modalidad se ordenan, de manera en

el árbol están contiguas.

Las fórmulas modales caja se procesan acorde a la Definición 9 después de identificar todas las ramas que deben generarse. Si se produce alguna contradicción en un nodo debido a este tipo de fórmulas, dicho nodo debe descartarse. No obstante, debe existir al menos un nodo hijo en cada rama, de lo contrario se procede a realizar *backtracking*. En la estructura sintáctica de árbol se observan nodos dibujados con una línea punteada, estos nodos se consideran alternativas y serán empleados si en un nivel más profundo es necesario realizar *backtracking*.

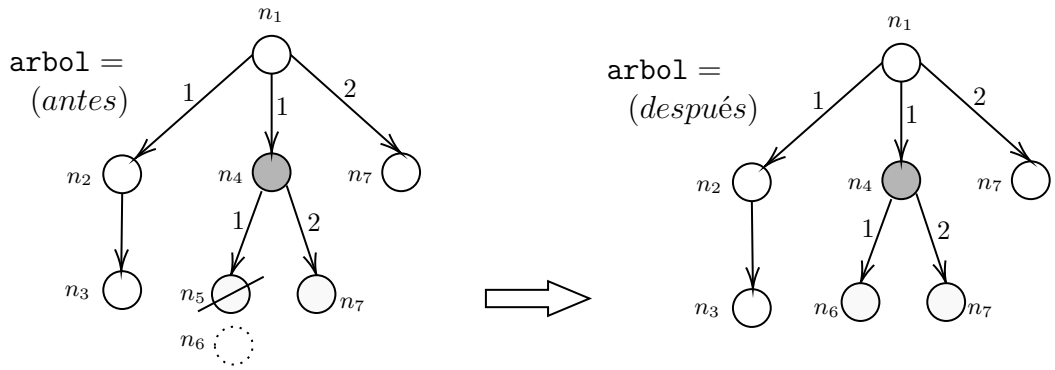
El algoritmo posee una optimización que tratará de fusionar ramas de la misma modalidad, de ser posible, para disminuir la complejidad de modelo. Para esos casos, se observa que hay un nodo especial n_5 que no contiene ninguna fórmula. Este posee una bandera llamada $esSD$ con un valor verdadero.

Por último, la función se encarga de unir un nodo al árbol por cada rama existente. Todos los nodos poseen una bandera adicional *principal* que permite identificar cuales ya están en el árbol ($principal = verdadero$) y cuales son alternativas ($principal = falso$). Por ejemplo, para determinar que n_3 y n_4 son nodos alternativas del nodo n_2.

En próximas iteraciones, cuando `nodoActual` $= n_5$ y se realice un llamado a la función `siguiente`, esta función determinará si es necesario conservar la rama donde se encuentra el nodo n_5. Si el conjunto de fórmulas de alguno de los nodos alternativos de n_5 es un subconjunto de las fórmulas del nodo de la rama derecha, considerando que ya se ha generado el subárbol de dicho nodo derecho, entonces la rama donde está n_5 se descarta de la estructura sintáctica de árbol. En caso contrario, se conserva la rama, eliminando n_5 y uniendo al árbol el próximo nodo alternativo de la rama, en este caso n_6.

Definición 12. La función `valNodosRep(árbol, nodosHijos)` = `hayNodosRep, nodosHijos` se define a continuación:

- `hayNodosRep` $\leftarrow false$.
- Verifica que todos los nodos en `nodosHijos` generados por la función `siguiente` no estén repetidos en el árbol con raíz `arbol`. Los nodos especiales ($\texttt{nH}^i_{esSD} = true$, $\texttt{nH}^i_{esSB} = true$) no se consideran.
- Si existe un nodo $\texttt{nH}^i$ repetido:
  - Si el nodo $\texttt{nH}^{i+1}$ no existe, retorna verdadero y una lista vacía.
  - Si $\texttt{nH}^i_{principal} = true$, `i+1` $\leq |$`nodosHijos`$|$ y $\texttt{nH}^{i+1}_{principal} = false$, entonces:
    - Se elimina el nodo $\texttt{nH}^i$ de la lista `nodosHijos`, $\texttt{nH}^{i+1}_{principal} = true$ y `hayNodosRep` $\leftarrow true$.
  - Si $\texttt{nH}^i_{principal} = false$, `i+1` $\leq |$`nodosHijos`$|$ y $\texttt{nH}^{i+1}_{principal} = false$, entonces:



$\texttt{nodoActual} = n_4$

$\texttt{nodosHijos} = [\{\overset{n_5}{\langle 1\rangle}\top,\ q,\ \overset{n_5}{\langle 1\rangle}E,\ \langle 1\rangle\langle 1\rangle E\},\ \{\overset{n_6}{\langle 1\rangle}\top,\ p,\ \overset{n_6}{\langle 1\rangle}E,\ \langle 1\rangle\langle 1\rangle E\},$
$\{\overset{n_7}{\langle 2\rangle}\top,\ q,\ \overset{n_7}{\langle 2\rangle}E,\ \langle 2\rangle\langle 1\rangle E\}]$

$\texttt{valNodosRep}(\texttt{arbol},\ \texttt{nodosHijos}) = [\{\overset{n_6}{\langle 1\rangle}\top,\ p,\ \overset{n_6}{\langle 1\rangle}E,\ \langle 1\rangle\langle 1\rangle E\},$
(Se actualizan los nodos en el árbol) $\{\overset{n_7}{\langle 1\rangle}\top,\ p,\ \overset{n_7}{\langle 1\rangle}E,\ \langle 1\rangle\langle 1\rangle E\}]$

Figura 3.6: Ejemplo del funcionamiento de `valNodosRep`.

  - Se elimina el nodo $\texttt{nH}^i$ de la lista `nodosHijos` y `hayNodosRep` $\leftarrow$ *true*.
- En caso contrario, retorna *true* y una lista vacía.

- En caso contrario, retorna `hayNodosRep` y `nodosHijos`.

Después de que se genera la lista `nodosHijos`, ante el llamado de la función `siguiente`, continua el proceso de llamar a la función `valNodosRep`.

Ejemplo 3.7. En la Figura 3.6 se observa un ejemplo cuando la lista `nodosHijos` posee dos nodos repetidos: n_5 y n_7. La función `valNodosRep` tendrá que eliminar alguno de los nodos repetidos, de ser posible. Se observa que en la rama donde se encuentra n_5 hay al menos un nodo alternativo, expresado con una línea punteada, por lo que se procederá a eliminar n_5 y el nodo n_6 será el nuevo nodo que estará unido al árbol en dicha rama. Posteriormente, se vuelve a verificar la presencia de nodos repetidos y se repite el proceso.

Cuando no existan nodos alternativos que puedan sustituir a un nodo repetido, el algoritmo terminará su ejecución, esto garantiza que no se cicle ya que es imposible continuar construyendo el árbol.

Definición 13. La función `vueltaAtrás(nodoActual, pilaNodos, backtrackNodos, árbol) = hayAlt, árbol` realiza una vuelta atrás siempre que sea necesario, con el objetivo de probar otras alternativas de estructuras sintácticas de árbol que permitan satisfacer

la fórmula. En el caso de que haya nodos en la lista `nodosHijos`, estos nodos sustituirán al nodo actual y sus alternativas o al nodo padre del nodo actual y sus alternativas, debido a casos especiales de modalidades inversas en operadores caja. En caso contrario, únicamente se toma el nodo en la cabeza de la lista `nodoActual` y realiza la vuelta atrás hasta el punto en el que ese nodo fue agregado a la estructura sintáctica de árbol.

- Si el tamaño de la lista `nodosHijos` es mayor que cero, entonces:
  - Se crea una variable auxiliar `vNodoActual` $\leftarrow$ `nodoActual`.
  - Se toma el primer nodo nPH en `nodosHijos` y se verifica si nPH_{padre} es distinto de $nodoActual_{padre}$, entonces:
    - Se establece `vNodoActual` $\leftarrow$ `nodoActual`$_{padre}$.
  - Se crea una lista de nodos llamada `nodosPadreO` $\leftarrow$ [`vNodoActual`] y una variable `nodoItera` $\leftarrow vNodoActual_{derecha}$. Mientras `nodoItera` sea distinto de un valor nulo, se agrega el nodo `nodoItera` y se modifica el valor: `nodoItera` $\leftarrow$ `nodoItera`$_{derecha}$.
  - Mientras la longitud de `pilaNodos` > 0 se verifica si el elemento en la cabeza de la pila está contenido en la lista `nodosPadreO`, de ser así se elimina de la pila. En caso contrario, el ciclo termina.
  - Por cada nodo `nS` en `nodosPadreO` se efectúa lo siguiente: si la longitud de la pila `backtrackNodos` > 0 y el nodo en la cabeza es igual a `nS`, entonces se remueve dicho nodo de `backtrackNodos`.
  - Cada nodo en la lista `nodosHijos` se agrega a la pila `pilaNodos`.
  - Si el valor de `vNodoActual`$_{padre}$ es distinto de nulo, es decir, el nodo es raíz y el nodo `vNodoActual` posee al menos un hijo, se realiza lo siguiente:
    - Dada la lista de nodos hijos `nHP` del nodo `vNodoActual`$_{padre}$ se procede a realizar una búsqueda hasta encontrar el primer nodo de la lista `nodosPadreO`. Una vez que se localiza, se almacena la posición del nodo en la variable auxiliar `iPH` y posteriormente uno a uno se remueven de `nHP`, hasta llegar al último nodo de la lista `nodosPadreO`. Finalmente, en la posición `iPH` se insertan todos los nodos contenidos en la lista `nodosHijos`.
  - La función retorna verdadero (`hayAlt`) y la variable `arbol` que corresponde a la raíz de la estructura sintáctica de árbol en construcción.
- En caso contrario:
  - Se crea una pila vacía llamada `pilaBusAlt` y una variable llamada `pNodoBack`. Si el parámetro `nodoActual` posee un valor nulo, entonces:
    - Se extrae el nodo `cabBT` de la cabeza de `backtrackNodos`.
    - Mientras el nodo en la cabeza de `pilaNodos` sea distinto de `cabBT` y la longitud de `pilaNodos` > 0, entonces se remueve el elemento en la cabeza de `pilaNodos`.

- Si `cabBT` es distinto del elemento en la cabeza de `pilaNodos`, entonces la función devuelve falso (`hayAlt`) y un valor nulo (`arbol`). En caso contrario se establece `pNodoBack` ← `cabBT`.

- Si `nodoActual` no corresponde a un valor nulo, se agrega `nodoActual` a la pila `pilaBusAlt`.
- Mientras la longitud de `pilaBusAlt` > 0, se realiza:
  - Se extrae el elemento en la cabeza de `pilaBusAlt` y se establece a la variable `pNodoBack`.
  - Si $\texttt{pNodoBack}_{esAlt} = true$ y $\texttt{pNodoBack}_{esSB} = false$, entonces:
    - Si $\texttt{nodoActual}_{padre}$ y $(\texttt{pNodoBack}_{izquierda})_{padre}$ son nodos raíces, es decir, son distintos de un valor nulo, en la lista de nodos hijos del nodo $(\texttt{pNodoBack}_{izquierda})_{padre}$ se busca el nodo $\texttt{pNodoBack}_{izquierda}$ y se remplaza por `pNodoBack`.
    - Si $\texttt{nodoActual}_{padre}$ es nodo raíz, pero $(\texttt{pNodoBack}_{izquierda})_{padre}$ no lo es, entonces se establece `arbol` ← `pNodoBack`.
    - Se establece la bandera `desact` a verdadero para el nodo `pNodoBack` así como a todos sus nodos hijos.
    - Se modifica $\texttt{pNodoBack}_{esAlt}$ con un valor de falso.
    - Si el nodo $\texttt{pNodoBack}_{padre}$ es raíz, entonces se actualiza `arbol` ← `pNodoBack`.
  - Si $\texttt{pNodoBack}_{esSB} = true$:
    - Se desactiva el nodo `pNodoBack` estableciendo la bandera `desact` a verdadero.
    - Si la longitud de `backtrackNodos` $= 0$, la función devuelve falso (`hayAlt`) y un valor nulo (`arbol`).
    - Se extraen los nodos de la pila `backtrackNodos` mientras el nodo en la cabeza esté deshabilitado, es decir, la bandera `desact` $= true$. Si la longitud de `backtrackNodos` $= 0$ después de la extracción, la función devuelve falso (`hayAlt`) y un valor nulo (`arbol`).
    - Mientras el nodo en la cabeza de `pilaNodos` sea distinto de `cabBT` y la longitud de `pilaNodos` > 0, entonces se remueve el elemento en la cabeza de `pilaNodos`.
    - Si `cabBT` es distinto del elemento en la cabeza de `pilaNodos`, entonces la función devuelve falso (`hayAlt`) y un valor nulo (`arbol`). En caso contrario agrega `cabBT` a la pila `pilaBusAlt`.
  - Si ninguna de las dos condiciones anteriores sucedió, la función retorna falso (`hayAlt`) y un valor nulo (`arbol`).
- La función regresa verdadero (`hayAlt`) y la variable `arbol`.

Ejemplo 3.8. Se observa en la Figura 3.7 un ejemplo de *backtracking* realizado por la función `vueltaAtrás`. Considere la fórmula ϕ_5, el conjunto $Lean(\phi_5)$, el `arbol` justo antes

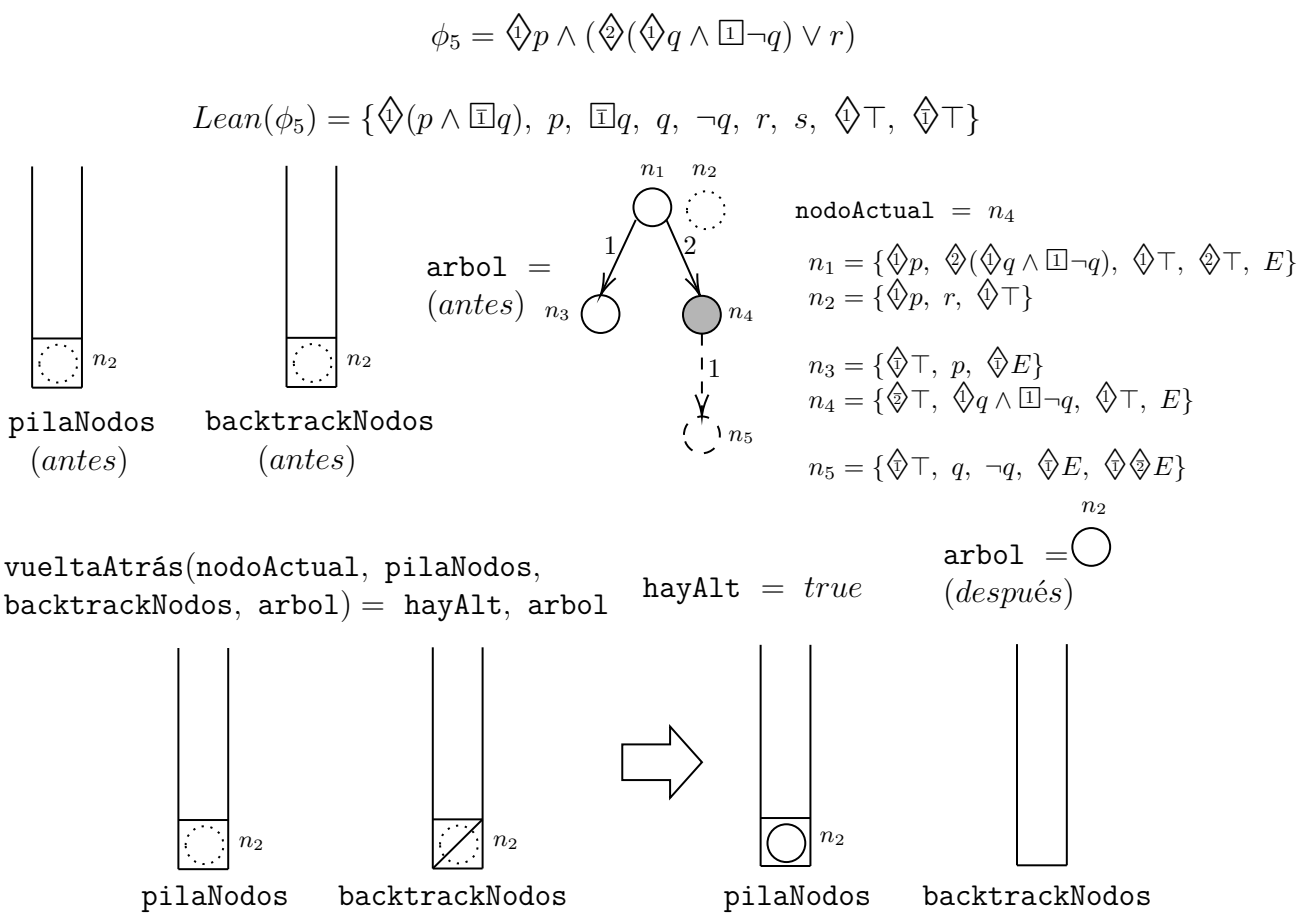


Figura 3.7: Ejemplo de ejecución de la función `vueltaAtrás` con ϕ_5 como entrada.

del *backtracking*, y las pilas `pilaNodos` y `backtrackNodos`.

En este caso se identifica una contradicción al momento de que el nodo n_4 es procesado por la función `siguiente` y no cuenta con nodos alternativos. Este nodo está dibujado con una línea punteada más prominente y etiquetado como n_5. La función `vueltaAtrás` se ejecuta y el primer paso que realiza es tomar el siguiente nodo de la pila `backtrackNodos`.

El nodo disponible para realizar *backtracking* es n_2 porque fue el que se extrajo de `backtrackNodos`. Posteriormente, se comienzan a extraer los nodos de la pila llamada `pilaNodos` hasta encontrar el nodo n_2. En este caso es el primero que resulta de extraer el primer nodo de `pilaNodos`.

Finalmente, el algoritmo sustituye en el árbol el nuevo nodo a probar n_2. Todas las ramas y nodos de niveles inferiores se eliminan, tomando como referencia dicho nodo. Lo que en el ejemplo se traduce en volver a comenzar desde un nodo raíz n_2. La función retorna la bandera `hayAlt` con un valor verdadero para indicar al algoritmo que puede continuar.

Ejemplo 3.9. En la Figura 3.8 se observa otro caso de *backtracking* adicional al mostrado

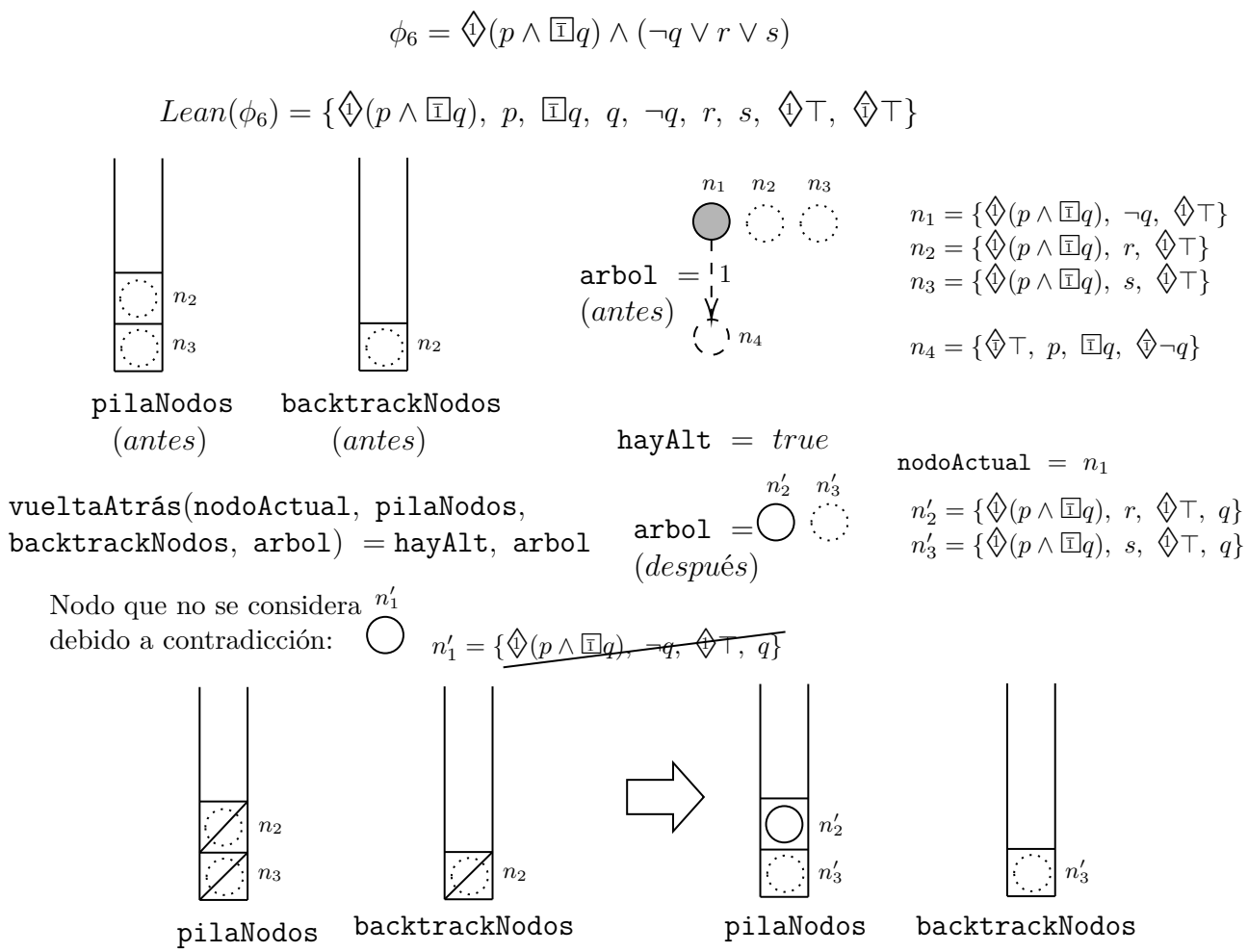


Figura 3.8: Ejemplo de ejecución de la función `vueltaAtrás` con ϕ_6 como entrada.

en el ejemplo anterior. Considere la fórmula ϕ_6, el conjunto $Lean(\phi_6)$, el `arbol` justo antes del *backtracking*, y las pilas `pilaNodos` y `backtrackNodos`.

En este caso, no se produce una contradicción directamente sino una inconsistencia en la estructura sintáctica de árbol. Esta se produce cuando la función `siguiente` ha recibido a n_1 y se intenta generar el único nodo hijo n_4. Se puede observar que existe una fórmula modal caja $[1]q$ contenida en n_4. En este caso, la función `siguiente` verifica que q se cumpla en el nodo padre n_1, debido a la modalidad inversa de la fórmula modal caja.

Se genera la inconsistencia, puesto que q no está contenida en el nodo padre de n_4. Entonces la función `vueltaAtrás` debe regenerar el nodo padre y las alternativas de dicho nodo padre. En este caso, basta con que q esté contenida en n_1, n_2 y n_3, por lo que procede a realizarlo. No obstante, al agregar q a n_1 e intentar construir el nodo n'_1 se produce una contradicción, como se puede observar en el ejemplo, por esta razón debe eliminarse.

En este proceso, el algoritmo obtiene los próximos nodos de la pila `backtrackNodos` hasta extraer por completo los nodos alternativos de n_1, así como en la pila llamada `pilaNodos`. Posteriormente, los nuevos nodos n_2' y n_3' se agregan a `pilaNodos` para que en la próxima iteración el algoritmo considere n_2'. Finalmente, se agrega n_3' a la pila `backtrackNodos` puesto que es un nodo alternativo.

Definición 14. La función `genAncesFórmulas` se encarga de generar fórmulas que se cumplirán en los nodos ancestros, como se detalla en los últimos dos puntos de la Definición 7.

En el algoritmo esta función recibe el conjunto `lean` y tiene acceso al árbol sintáctico de la fórmula de entrada, así como al `nivelActual` del árbol. Comienza a generar en cada llamado las fórmulas que podrán estar contenidas en los nodos y que se satisfacen en nodos padres, abuelos, etc. Se realiza un llamado conforme se va incrementando el número de niveles de la estructura sintáctica de árbol, es decir, un llamado por cada nivel, por lo que sin importar si se realiza una vuelta atrás estas fórmulas permanecen sin cambios ya que se consideran todas las posibilidades al momento de generar las fórmulas.

A partir del árbol sintáctico de la fórmula de entrada ϕ se realiza el siguiente proceso:

- Si no se ha creado `ancesLNodos`, se crea una lista vacía `ancesLNodos`, la cual almacenará listas de nodos.
- Si la lista `respaldoAFormu` no se ha construido, se le asigna una lista vacía, que guardará una lista de listas de nodos.
- Si la lista `ancesFórmulas` no existe, se establece `ancesFórmulas` $\leftarrow [\,]$.
- Se obtienen las fórmulas que deben estar presentes en el nivel uno (nodos raíces) acorde con la Definición 9, $\Vdash \phi$, la lista de nodos generados se llamará `nodosG`.
- Si `ancesLNodos` es vacía
  - Por cada nodo en `nodosG` se realiza lo siguiente:
    - Se extraen las fórmulas modales diamante presentes en el nodo y se obtiene la modalidad inversa n.
    - Se genera una nueva fórmula $\psi_0 = \diamondsuit_n(p_0 \wedge \ldots \wedge p_i)$ de manera que p_i corresponde a las variables proposicionales presentes en el nodo.
  - Todas las fórmulas generadas en el paso anterior se almacenan en una lista y se agregan a `ancesFórmulas`.
  - Se debe efectúa un respaldo de `ancesLNodos` que será almacenado en la lista `respaldoAFormu`.
- En caso contrario, si la longitud de `ancesLNodos` > 0:

- Se debe realizar un respaldo de `ancesLNodos` que será almacenado en la lista `respaldoAFormu`.
- Se eliminan las variables proposicionales contenidas en cada nodo de cada lista de `ancesLNodos` y se eliminan nodos repetidos en cada lista presente en `ancesLNodos`, el objetivo es avanzar al siguiente nivel de profundidad de entre las estructuras sintácticas de árbol.
- Se define la variable `rvueltaAtrás` $\leftarrow false$.
- Mientras no se haya generado la lista de fórmulas correspondiente al nivel actual ($\texttt{ancesFórmulas}_{\texttt{nivelActual}}$):
  - Si se realizó una vuelta atrás se eliminan las variables proposicionales contenidas en cada nodo de cada lista de `ancesLNodos` y se eliminan nodos repetidos en cada lista presente en `ancesLNodos`.
  - Si el `nivelActual` indica que la función se encuentra en un nivel distinto del raíz:
    1. Por cada lista de nodos en `ancesLNodos`, se verifica si existe un caso: cuando en un nodo existe una fórmula testigo $\Diamond\!\!\!\!\!\!\diamond_n \top$ y una fórmula $\psi_c = \boxed{n}\psi_{c2}$, entonces se extrae la subfórmula ψ_{c2} y acorde con la Definición 9 se realiza $\Vdash \psi_{c2}$. Esto generará una lista de nodos con fórmulas que deberán satisfacerse en el nodo padre por cada rama `rLF`. Se crea una lista `LF` que tendrá almacenado el producto cruz de `LF` con la lista de nodos `rLF`, como si de conjuntos se tratase.
    2. Si la lista `rLF` no es vacía, se verifica que las fórmulas de cada lista de nodos generados en `rLF` estén presentes en su nodo padre. El nodo padre está almacenado en un nivel superior del árbol que está contenido en `respaldoAFormu` en la posición `nivelActual` $- 1$, se verifica que dicho nodo padre genere al nodo hijo. Si no están presentes, en el nodo padre se agregan las fórmulas de `rLF` y se habilita la bandera `rVueltaAtrás` $\leftarrow true$, en caso contrario se mantiene `rVueltaAtrás` $\leftarrow false$.
    3. Si `rVueltaAtrás` $= true$, se actualiza la variable `nivelActual` $\leftarrow$ `nivelActual` $- 1$, `ancesLNodos` $\leftarrow \texttt{respaldoAFormu}_{\texttt{nivelActual}}$ y se genera una nueva fórmula $\psi_0 = \Diamond\!\!\!\!\!\!\diamond_n(p_0 \wedge \ldots \wedge p_i)$ de manera que p_i corresponde a las variables proposicionales presentes en cada nodo padre afectado, de manera que cada fórmula nueva se agrega a la lista $\texttt{ancesLNodos}_{\texttt{nivelActual}}$.
  - Si `rVueltaAtrás` $= false$, por cada lista de nodos en `ancesLNodos`, que representa los posibles nodos de una sola rama del nivel actual, se efectúa lo siguiente:
    - Se agrupan por un lado las fórmulas modales diamante en listas individuales, separado por modalidades, y se almacenan en `formuDiam`, y por otro lado las fórmulas modales caja, también en listas individuales separado por modalidades que serán guardadas en la lista `formuCaja`.

Considerando cada fórmula modal diamante, se extrae la modalidad y sin repetirlas se agregan a la lista `modDiam`.

- ◇ Se crea una lista vacía nombrada `appCaja`.
- ◇ Por cada lista contenida en `formuCaja`, que contiene fórmulas modales de la forma $\boxed{n}\psi_{c3}$, se obtienen las fórmulas que deben estar presentes en un nodo, dada la Definición 9, realizando $\Vdash \psi_{c3}$. Se fusionan todos los nodos obtenidos de una misma lista de fórmulas modales caja, si es que no hay contradicciones, y se agrega el nodo resultante a `appCaja`.
- ◇ Por cada lista almacenada en `formuDiam`, que posee fórmulas modales de la forma $\diamondsuit_{n}\psi_{d3}$, se extraen las fórmulas que deben estar presentes en un nodo, dada la Definición 9, realizando $\Vdash \psi_{d3}$. Cada nodo, perteneciente a una misma lista de fórmulas modales diamante, se almacena en una lista llamada `mAppDiam` y se realiza lo siguiente: para cada elemento de `mAppDiam` se crea un nuevo nodo que fusione $\texttt{mAppDiam}_i$ y cada posibilidad del conjunto potencia de todos los nodos subsecuentes de $\texttt{mAppDiam}_i$, finalmente cada nuevo nodo se almacena en `appDiam`.
- ◇ Después de realizar los dos pasos anteriores se tiene una lista de nodos `appCaja` y una lista de listas de nodos llamada `appDiam`, donde cada lista corresponde a una rama de nodos hijos que origina cada fórmula de una misma modalidad del tipo diamante.
- ◇ Por cada nodo `aNC` en `appCaja` que fue originado por fórmulas modales caja de modalidad n, se procede a identificar la lista `lAD` en `appDiam` que haya sido originada por fórmulas modales diamante de modalidad n y posteriormente, se agregan las fórmulas del nodo `aNC` a cada nodo de la lista `lAD`.
- ◇ Cualquier nodo repetido, sobre la misma lista, o que posea contradicciones con respecto a sus variables proposicionales se remueve de `appDiam`.
- ◇ Considera cada lista `lAD` de nodos contenida en `appDiam` y cada nodo existente en dicha lista para realizar lo siguiente: se genera una nueva fórmula $\psi_0 = \diamondsuit_{n}(p_0 \wedge \ldots \wedge p_i)$ de manera que p_i corresponde a las variables proposicionales presentes en ese nodo, tal que n es la modalidad de las fórmulas modales diamante que originaron la lista `lAD`.
- ◇ Todas las fórmulas generadas en el paso anterior se almacenan en una lista y se agregan a `ancesFórmulas`.
- ◇ Se establece `ancesLNodos` $\leftarrow$ [ ] y cada lista de nodos en `appDiam` se agrega a `ancesLNodos`.
- ◇ En este punto se repiten los pasos enumerados anteriormente en el mismo orden.
- ◇ Si `rVueltaAtrás` $= false$, entonces se realiza lo siguiente: considere cada modalidad n_1 contenida en `modDiam` y cada fórmula ψ_0 tomada

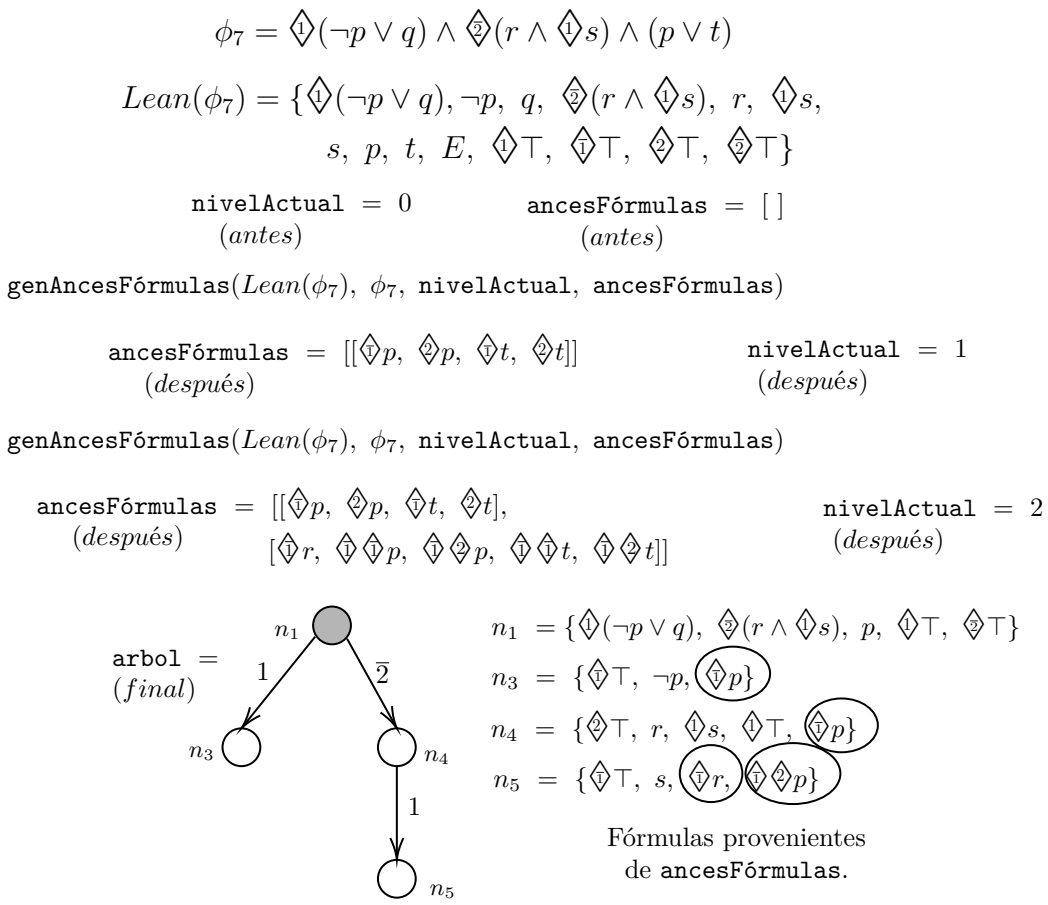


Figura 3.9: Ejemplo de la salida de la función `genancesFórmulas` ante una fórmula ϕ_7.

de la lista `ancesFórmulas`, se procede a crear una nueva fórmula $\psi_1 = \langle u\rangle\psi_0$ y agregarla a `ancesFórmulas` en la posición `nivelActual`.

Ejemplo 3.10. En la Figura 3.9 se observa una fórmula ϕ_7 y el conjunto $Lean(\phi_7)$. Inicialmente, la variable `nivelActual` empieza en cero, puesto que es cuanto el algoritmo comienza a generar nodos raíces. Adicionalmente, `ancesFórmulas` es una lista vacía que almacenará la salida de la función `genAncesFórmulas`. En el momento en el que la función `siguiente` recibe el primer nodo raíz, la función `genAncesFórmulas` se ejecuta internamente.

Después de la ejecución se observan que se ha generado una lista con 4 fórmulas modales diamante. Las fórmulas se construyen con la conjunción de las variables proposicionales de cada nodo del `nivelActual`. En este caso, habría solamente dos posibles nodos. En el caso de n_1 contiene únicamente a p y su nodo alternativo contendría únicamente a t.
Una vez que se tienen las fórmulas anteriores, es necesario concatenar a cada fórmula construida un operador modal diamante con la modalidad inversa de todas las fórmulas modales diamante contenidas en n_1. En este caso, sólo hay dos operadores modales que

$$\phi_8 = \langle 1 \rangle(\langle \overline{1} \rangle q \vee \langle \overline{1} \rangle(r \wedge \langle 2 \rangle s)) \wedge \neg q \wedge [2](\langle 1 \rangle \neg s \wedge \langle 3 \rangle t)$$

$$Lean(\phi_8) = \{\langle 1 \rangle(\langle \overline{1} \rangle q \vee \langle \overline{1} \rangle(r \wedge \langle 2 \rangle s)),\ \langle \overline{1} \rangle q,\ q,\ \langle \overline{1} \rangle(r \wedge \langle 2 \rangle s),\ r,\ \langle 2 \rangle s,\ s,\ \neg q,\ [2](\langle 1 \rangle \neg s \wedge \langle 3 \rangle t),$$
$$\langle 1 \rangle \neg s,\ \neg s,\ \langle 3 \rangle t,\ t,\ E,\ \langle 1 \rangle \top,\ \langle \overline{1} \rangle \top,\ \langle 2 \rangle \top,\ \langle \overline{2} \rangle \top,\ \langle 3 \rangle \top,\ \langle \overline{3} \rangle \top\}$$

Salida del algoritmo:

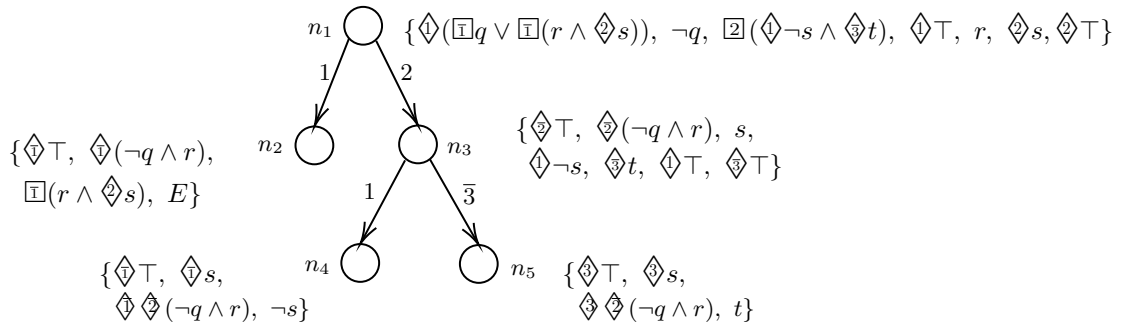


Figura 3.10: Estructura sintáctica de árbol $\mathfrak{A}_1$ que retorna el algoritmo cuando su entrada es ϕ_8.

serán concatenados: $\langle \overline{1} \rangle$ y $\langle \overline{2} \rangle$. Esto genera las cuatro fórmulas.

Finalmente, la variable `nivelActual` se incrementa. Se observa que `ancesFórmulas` es una lista de listas, debido a que se genera una lista cada vez que se llama a la función y se agrega a `ancesFórmulas`.

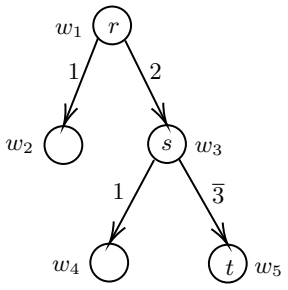
La siguiente llamada a `ancesFórmulas` se realiza cuando la función `siguiente` recibe a n_3, que corresponde a el primer nodo del nivel 1 en la estructura sintáctica de árbol. Nuevamente, se repite el mismo proceso. En n_3 no vay variables proposicionales sin negación. Por otro lado en n_4 está r. Dadas las fórmulas modales diamante de n_3 y n_4 se considera únicamente la modalid: $\langle \overline{1} \rangle$. De allí surge la fórmula $\langle \overline{1} \rangle r$ presente en `ancesFórmulas`.

Finalmente, cuando el `nivelActual` es uno o mayor, la función `genAncesFórmulas` toma las fórmulas generadas del nivel previo y concatena las modalidades consideradas para este nivel. Por esta razón es que se construye, por ejemplo, la fórmula $\langle \overline{1} \rangle \langle \overline{1} \rangle p$, debido a que proviene de concatenar $\langle \overline{1} \rangle$ con $\langle \overline{1} \rangle p$.

A continuación se muestran algunos ejemplos del funcionamiento del algoritmo, para ello se emplearán fórmulas que se satisfacen. Se analizarán entradas y salidas de un par de funciones esenciales, además de presentar la estructura sintáctica de árbol resultante y una estructura de Kripke con forma de árbol que satisface a la fórmula y que se puede derivar fácilmente de la salida del algoritmo.

Ejemplo 3.11. En la Figura 3.10 se muestra la fórmula ϕ_8 y la estructura sintáctica de

$$\phi_8 = \langle\!\langle 1 \rangle\!\rangle(\boxed{\overline{1}}q \vee \boxed{\overline{1}}(r \wedge \langle\!\langle 2 \rangle\!\rangle s)) \wedge \neg q \wedge \boxed{2}(\langle\!\langle 1 \rangle\!\rangle \neg s \wedge \langle\!\langle \overline{3} \rangle\!\rangle t)$$



$$\begin{aligned} W &= \{w_1,\ w_2,\ w_3,\ w_4, w_5\} \\ \mathcal{R} &= \{R_1, R_2, R_{\overline{3}}, R_{\overline{1}}, R_{\overline{2}}, R_3\} \\ R_1 &= \{(w_1, w_2), (w_3, w_4)\} \\ R_{\overline{1}} &= \{(w_2, w_1), (w_4, w_3)\} \\ R_2 &= \{(w_1, w_3)\} \\ R_{\overline{2}} &= \{(w_3, w_1)\} \\ R_3 &= \{(w_5, w_3)\} \\ R_{\overline{3}} &= \{(w_3, w_5)\} \end{aligned} \qquad \begin{aligned} V(r) &= \{w_1\} \\ V(s) &= \{w_3\} \\ V(t) &= \{w_5\} \\ V(q) &= \{\ \} \end{aligned}$$

Figura 3.11: Estructura de Kripke $\mathfrak{M}_3$ que satisface ϕ_8.

árbol que retorna el algoritmo ante esta entrada. Cada nodo de la estructura sintáctica de árbol posee un conjunto de fórmulas que se muestran a un costado de cada nodo.

Se puede observar que la subfórmula $\boxed{\overline{1}}q$ origina una contradicción debido a que deberá estar presente q en el nodo raíz, y ya está presente la fórmula $\neg q$. En este caso el algoritmo intenta construir un nodo hijo alternativo que contiene $\boxed{\overline{1}}(r \wedge \langle\!\langle 2 \rangle\!\rangle s)$, el cual es originado debido a la disyunción de la subfórmula $\boxed{\overline{1}}q \vee \boxed{\overline{1}}(r \wedge \langle\!\langle 2 \rangle\!\rangle s)$.

Finalmente, se puede apreciar que al existir dos fórmulas con modalidad diamante, por ejemplo, en el nodo n_1 o n_3 el algoritmo debe crear dos ramas independientes. Si no se hubiera generado una contradicción con $\boxed{\overline{1}}q$ el nodo n_1 tendría una única rama.

Considerando la estructura sintáctica de árbol $\mathfrak{A}_1$ que retorna el algoritmo frente a la fórmula de entrada ϕ_8, es posible construir una estructura de Kripke que satisfaga ϕ_8, como se muestra en la Figura 3.11. Esta propiedad se conoce como coherencia y en este trabajo está incluido en la prueba de corrección del algoritmo en el Teorema 2.

Se puede visualizar que en la estructura de Kripke ya no es necesario mantener las negaciones de variables proposicionales ni la variable proposicional comodín E, debido a que la función de valuación V retorna aquellos mundos donde se cumple cierta variable proposicional. En caso de que en un mundo no se cumpla ninguna fórmula, la función V para todas las variables proposicionales retornará conjuntos que no contendrán a dicho mundo.

Ejemplo 3.12. A continuación se describe brevemente otro ejemplo ante la fórmula de entrada ϕ_9 y la salida del algoritmo, mostrado en la Figura 3.12. En este ejemplo, la estructura sintáctica de árbol posee 5 niveles de profundidad. La fórmula de entrada está compuesta mayormente por fórmulas modales diamante con el objetivo de forzar que el árbol tenga una anchura y profundidad mayor.

$$\phi_9 = \langle\!\langle 1\rangle\!\rangle((q \wedge r) \wedge (\langle\!\langle 1\rangle\!\rangle p \wedge \langle\!\langle 3\rangle\!\rangle(q \wedge (\langle\!\langle 2\rangle\!\rangle r \wedge [\![2]\!](\langle\!\langle 2\rangle\!\rangle \neg t)))))$$

$$Lean(\phi_9) = \{\langle\!\langle 1\rangle\!\rangle((q \wedge r) \wedge (\langle\!\langle 1\rangle\!\rangle p \wedge \langle\!\langle 3\rangle\!\rangle(q \wedge (\langle\!\langle 2\rangle\!\rangle r \wedge [\![2]\!](\langle\!\langle 2\rangle\!\rangle \neg t))))),\ q,\ r,\ \langle\!\langle 1\rangle\!\rangle p,\ p,\ \langle\!\langle 3\rangle\!\rangle(q \wedge (\langle\!\langle 2\rangle\!\rangle r \wedge [\![2]\!](\langle\!\langle 2\rangle\!\rangle \neg t))),$$
$$q,\ \langle\!\langle 2\rangle\!\rangle r,\ r,\ [\![2]\!](\langle\!\langle 2\rangle\!\rangle \neg t),\ \langle\!\langle 2\rangle\!\rangle \neg t,\ \neg t,\ E,\ \langle\!\langle 1\rangle\!\rangle \top,\ \langle\!\langle 1\rangle\!\rangle \top,\ \langle\!\langle 2\rangle\!\rangle \top,\ \langle\!\langle 3\rangle\!\rangle \top\}$$

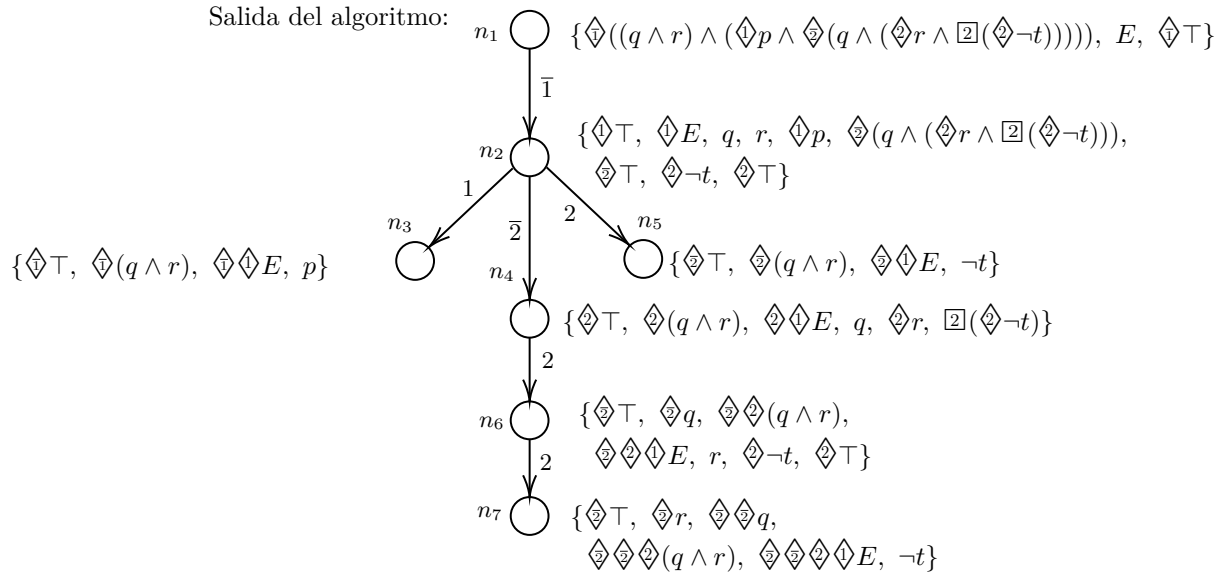


Figura 3.12: Estructura sintáctica de árbol $\mathfrak{A}_2$ que retorna el algoritmo cuando su entrada es ϕ_9.

Si se observa el nodo n_4 se aprecia la presencia de la fórmula $[\![2]\!](\langle\!\langle 2\rangle\!\rangle \neg t)$. Esta fórmula origina, acorde a la Definición 9, que deba estar presente la fórmula $\langle\!\langle 2\rangle\!\rangle \neg t$ en todos los nodos vecinos a los que se puede llegar mediante la modalidad 2, debido a la semántica del operador modal caja. Como se permiten inversas, la fórmula $\langle\!\langle 2\rangle\!\rangle \neg t$ deberá estar contenida tanto en n_6 como en n_2.

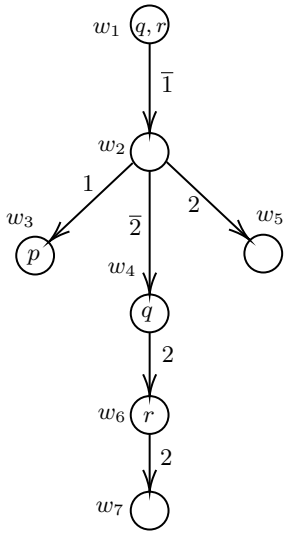
Finalmente, al igual que en el ejemplo anterior, a partir de la estructura sintáctica de árbol se puede generar una estructura de Kripke que satisface la fórmula, la cual se observa en la Figura 3.13.

Prueba de corrección

Con el objetivo de realizar la prueba de corrección del algoritmo se deberá demostrar los conceptos de coherencia y completitud, los cuales se detallan a continuación.

Teorema 2. (Coherencia). Si el algoritmo recibe una fórmula ϕ y retorna verdadero, además de una estructura sintáctica de árbol $\mathfrak{A} = (raíz, \mathfrak{A}_1, \ldots, \mathfrak{A}_i)$, entonces ϕ se satisface bajo una estructura de Kripke $\mathfrak{M} = (W, \mathcal{R}, V)$ en un nodo $w \in W$. Es decir, $raíz \Vdash \phi$ implica $\mathfrak{M}, w \models \phi$

$$\phi_9 = \langle\!\langle\overline{1}\rangle\!\rangle((q \wedge r) \wedge (\langle\!\langle 1\rangle\!\rangle p \wedge \langle\!\langle\overline{2}\rangle\!\rangle(q \wedge (\langle\!\langle 2\rangle\!\rangle r \wedge [\![2]\!](\langle\!\langle\overline{2}\rangle\!\rangle\neg t)))))$$



$$\begin{aligned}
W &= \{w_1,\ w_2,\ w_3,\ w_4, w_5, w_6, w_7\}\\
\mathcal{R} &= \{R_1, R_2, R_{\overline{1}}, R_{\overline{2}}\}\\
R_1 &= \{(w_2, w_3), (w_2, w_1)\}\\
R_{\overline{1}} &= \{(w_3, w_2), (w_1, w_2)\}\\
R_2 &= \{(w_2, w_5), (w_4, w_6),\\
&\qquad (w_6, w_7), (w_4, w_2)\}\\
R_{\overline{2}} &= \{(w_5, w_2), (w_6, w_4), \}\\
&\qquad (w_7, w_6), (w_2, w_4)\}
\end{aligned}$$

$$\begin{aligned}
V(r) &= \{w_2, w_6\}\\
V(p) &= \{w_3\}\\
V(t) &= \{\ \}\\
V(q) &= \{w_2, w_4\}
\end{aligned}$$

Figura 3.13: Estructura de Kripke $\mathfrak{M}_4$ que satisface ϕ_9.

Prueba. La prueba del Teorema 2 se realiza mediante inducción estructural sobre la fórmula de entrada ϕ, considerando que a partir de la estructura sintáctica de árbol que retorna el algoritmo se puede construir una estructura de Kripke que satisface la misma fórmula de entrada del algoritmo. Esta prueba es constructiva en el sentido de que se expresa paso a paso como generar la estructura de Kripke.

Casos base.

- La fórmula ϕ es una variable proposicional p. El algoritmo devuelve verdadero y una estructura sintáctica de árbol $\mathfrak{A}$ con un único nodo: $(raíz)$, tal que $p \in raíz$. La estructura sintáctica de árbol $\mathfrak{A}$ se puede convertir en una estructura de Kripke $\mathfrak{M} = (W, \mathcal{R}, V)$, de manera que $W = \{raíz\}$, $\mathcal{R} = \{\}$ y $V(p) = \{raíz\}$. La fórmula ϕ se sostiene en $\mathfrak{M}$.

- La fórmula ϕ tiene la forma $\neg p$. El algoritmo devuelve verdadero y una estructura sintáctica de árbol $\mathfrak{A}$ con un solo nodo: $(raíz)$, tal que $\neg p \in raíz$. La estructura sintáctica de árbol $\mathfrak{A}$ se puede convertir en una estructura de Kripke $\mathfrak{M} = (W, \mathcal{R}, V)$, de manera que $W = \{raíz\}$, $\mathcal{R} = \{\}$ y $V(p) = \{\}$, entonces la fórmula ϕ se mantiene en $\mathfrak{M}$.

Paso inductivo.

- La fórmula es una disyunción: $\phi = \psi_1 \vee \psi_2$. El algoritmo devuelve una estructura sintáctica de árbol $\mathfrak{A}$ de modo que en el nodo $raíz$ se cumple $raíz \Vdash \phi$. Se puede afirmar que $raíz \Vdash \psi_1$ o $raíz \Vdash \phi_2$. Por hipótesis de inducción existe una estructura de Kripke $\mathfrak{M}'$ que satisface a algunas de las dos fórmulas: $\mathfrak{M}', w' \models \psi_1$ o $\mathfrak{M}', w' \models$

ψ_2. Por lo que se puede concluir que en $\mathfrak{M}'$ se sostiene $\phi = \psi_1 \vee \psi_2$, es decir $\mathfrak{M}'$, $w' \models \phi$.

- La fórmula contiene un operador de conjunción: $\phi = \psi_1 \wedge \psi_2$. Se sostiene que $raíz \Vdash \phi$. Por lo que las fórmulas ψ_1 y ψ_2 de manera independiente mantienen $raíz \Vdash \psi_1$ y $raíz \Vdash \psi_2$ verdaderas. Al aplicar la hipótesis de inducción se obtiene una única estructura de Kripke $\mathfrak{M}'$ que satisface a cada fórmula ψ_1 y ψ_2, debido a que la estructura sintáctica de árbol del que se genera la estructura con la hipótesis de inducción es la misma: $\mathfrak{M}'$, $w' \models \psi_1$ y $\mathfrak{M}'$, $w' \models \psi_2$. En consecuencia, la estructura $\mathfrak{M}'$ satisface $\psi_1 \wedge \psi_2$.

- La fórmula posee un operador modal de posibilidad: $\phi = \diamondsuit_m \psi$. El algoritmo retorna una estructura sintáctica de árbol $\mathfrak{A} = (raíz, \mathfrak{A}_1, \ldots, \mathfrak{A}_i)$. En algún subárbol $\mathfrak{A}' \in (\mathfrak{A}_1, \ldots, \mathfrak{A}_i)$ relacionado mediante la modalidad m, es decir, en el nodo raíz de $\mathfrak{A}'$ ($raíz_{\mathfrak{A}'}$) debe estar incluida la fórmula $\diamondsuit_m \top$, se cumple $raíz_{\mathfrak{A}'} \Vdash \psi$. Por hipótesis de inducción se obtiene una estructura de Kripke $\mathfrak{M}' = (W', \mathcal{R}', V')$ donde se mantiene ψ en $w' \in W'$.

  Es posible construir una estructura de Kripke $\mathfrak{M} = (W, \mathcal{R}, V)$ de manera que el conjunto de mundos se define como: $W = W' \cup \{raíz\}$; la familia de relaciones: $\mathcal{R} = \mathcal{R}'$, tal que $R'_m = R'_m \cup \{(raíz, raíz_{\mathfrak{A}'})\}$ y $R'_{\overline{m}} = R'_{\overline{m}} \cup \{(raíz_{\mathfrak{A}'}, raíz)\}$; y la función de valuación: $V = V'$, de modo que para todas las variables proposicionales $p \in raíz$ se considera $V'(p) = V'(p) \cup \{raíz\}$. Dicha estructura $\mathfrak{M}$ satisface a la fórmula ϕ en el nodo $raíz$.

- La fórmula está compuesta por un operador modal de necesidad: $\phi = \boxed{m} \psi$.

  - El algoritmo devuelve una estructura sintáctica de árbol $\mathfrak{A} = (raíz)$ con un único nodo, de manera que $raíz \Vdash \phi$. Se genera una estructura de Kripke $\mathfrak{M} = (W, \mathcal{R}, V)$ con $W = \{raíz\}$, $\mathcal{R} = \{\}$ y $V(p) = \{\}$. La fórmula ϕ se satisface en $\mathfrak{M}$.
  - El algoritmo devuelve una estructura sintáctica de árbol $\mathfrak{A} = (raíz, \mathfrak{A}_1, \ldots, \mathfrak{A}_i)$, con más de un nodo, donde se mantiene $raíz \Vdash \phi$. En todos las subestructuras sintácticas de árbol $\mathfrak{A}'_n \in (\mathfrak{A}_1, \ldots, \mathfrak{A}_i)$ que posean la fórmula $\diamondsuit_m \top$ en su nodo raíz ($raíz_{\mathfrak{A}'_n}$), se cumple $raíz_{\mathfrak{A}'_n} \Vdash \psi$. Por hipótesis de inducción cada estructura sintáctica de árbol $\mathfrak{A}'_n$ se puede transformar en una estructura de Kripke $\mathfrak{M}'_n$ que satisface a la fórmula ψ. De manera que la construcción de la estructura de Kripke $\mathfrak{M} = (W, \mathcal{R}, V)$ que satisface a la fórmula ϕ se define a continuación.

    De manera inicial se considera el mismo nodo raíz de $\mathfrak{A}$ en la estructura de Kripke $\mathfrak{M}$, por lo que $W = \{raíz\}$. Además, para todas las variables proposicionales $p \in raíz$ se debe considerar $V(p) = \{raíz\}$.

    Posteriormente, por cada estructura $\mathfrak{M}'_n$ se realiza lo siguiente: el conjunto de mundos se define como $W = W \cup W'_n$; la familia de relaciones $\mathcal{R} = \mathcal{R} \cup R'_n$ de manera que $R'_{m,n} = R'_{m,n} \cup \{(raíz, raíz_{\mathfrak{A}'_n})\}$ y $R'_{\overline{m},n} = R'_{\overline{m},n} \cup \{(raíz_{\mathfrak{A}'_n}, raíz)\}$;

la función de valuación se describe como: para todas las variables proposicionales $q \in \phi$, $V(q) = V(q) \cup V'_n(q)$. Así, en la estructura $\mathfrak{M}$ se mantiene la fórmula ϕ.

Por consiguiente, queda demostrado el Teorema 2 relacionado a la coherencia del algoritmo. Para probar la completitud se requiere de una definición adicional, la cual permita convertir cualquier estructura de Kripke en una estructura sintáctica de árbol y mantener la satisfacción de la fórmula.

Teorema 3. Si existe una estructura de Kripke $\mathfrak{M}$ que satisface a una fórmula ϕ en un mundo w de dicho modelo, entonces existe una estructura sintáctica de árbol $\mathfrak{A} = (r,\ \mathfrak{A}_1,\ \mathfrak{A}_2,\ \ldots,\ \mathfrak{A}_n)$ en donde se cumple $r \Vdash \phi$.

Prueba. La estructura de Kripke $\mathfrak{M}$ se puede convertir en una estructura que posee la propiedad de modelo de árbol $\mathfrak{M}_\mathfrak{t}$ empleando el Teorema 1 (Propiedad de modelo de árbol). La fórmula de entrada ϕ está en su forma normal negativa, sin pérdida de generalidad empleando la Definición 2. El conjunto *lean* se genera dada la fórmula ϕ, como se especifica en la Definición 6.

Casos base.

- La fórmula tiene la forma: $\phi = p$. Existe una estructura $\mathfrak{M}_t = (W, \mathcal{R}, V)$, tal que $W = \{w\}$, $\mathcal{R} = \{\ \}$ y $V(p) = \{w\}$. Con base en dicha estructura es posible crear una estructura sintáctica de árbol $\mathfrak{A} = (r)$ de manera que el conjunto de fórmulas de $r = \{p\}$ y $p \in \mathit{lean}$. Se sostiene $r \Vdash \phi$.
- Si la fórmula ϕ es la negación de una variable proposicional $\neg p$, entonces existe una estructura $\mathfrak{M}_t = (W, \mathcal{R}, V)$ con $W = \{w\}$, $\mathcal{R} = \{\ \}$ y $V(p) = \{\}$. Con base en $\mathfrak{M}_t$ se crea la estructura sintáctica de árbol $\mathfrak{A} = (r)$, tal que las fórmulas en dicho nodo son: $\{\neg p\}$ y $\neg p \in \mathit{lean}$. Se cumple $r \Vdash \phi$.

Paso inductivo.

- La fórmula contiene un operador de disyunción: $\phi = \psi_1 \vee \psi_2$, entonces existe una estructura $\mathfrak{M} = (W, \mathcal{R}, V)$ que satisface a ϕ en el nodo $w \in W$. Dicha estructura satisface a ψ_1 o a ψ_2, es decir, se cumple que $\mathfrak{M}, w \models \psi_1$ o $\mathfrak{M}, w \models \psi_2$, por lo que por hipótesis de inducción existe una estructura sintáctica de árbol $\mathfrak{A}$ de manera que en el nodo r se sostiene $r \Vdash \psi_1$ o $r \Vdash \psi_2$. En conclusión en la estructura sintáctica de árbol $\mathfrak{A}$ se cumple $r \Vdash \phi$.
- La fórmula es una conjunción: $\phi = \psi_1 \wedge \psi_2$. Existe una estructura $\mathfrak{M} = (W, \mathcal{R}, V)$ que satisface a ϕ en $w \in W$. En la estructura $\mathfrak{M}$ se satisfacen de manera disyunta bajo el mismo nodo las fórmulas ψ_1 y ψ_2, es decir, se cumple que $\mathfrak{M}, w \models \psi_1$ y $\mathfrak{M}, w \models \psi_2$. Al aplicar la hipótesis de inducción existe una estructura sintáctica de árbol $\mathfrak{A}$ con nodo raíz r tal que $r \Vdash \psi_1$ y $r \Vdash \psi_2$. En consecuencia, en la estructura sintáctica de árbol $\mathfrak{A}$ se cumple $r \Vdash \phi$.

- Para el caso modal de posibilidad: $\phi = \diamondsuit_m \psi$. Una estructura $\mathfrak{M}_t = (W, \mathcal{R}, V)$ satisface a ϕ en $w \in W$. En algún nodo h con $(w, h) \in R_m$ se sostiene ψ. Por hipótesis de inducción existe una estructura sintáctica de árbol $\mathfrak{A}' = (r', \mathfrak{A}'_1, \ldots, \mathfrak{A}'_i)$, tal que $r' \Vdash \psi$. Se crea un nuevo nodo $raíz = \{\diamondsuit_m \top, E\}$. Al conjunto de fórmulas de r' se le agregan las fórmulas $\diamondsuit_m \top$ y $\diamondsuit_m E$. Se construye la estructura sintáctica de árbol $\mathfrak{A} = (raíz, \mathfrak{A}')$ y en el nodo raíz de $\mathfrak{A}$ se cumple $raíz \Vdash \phi$.

- La fórmula tiene la forma: $\phi = \boxed{m}\psi$.

  - La estructura $\mathfrak{M}_t$ donde se satisface ϕ posee un único mundo w, es decir $\mathfrak{M}_t, w \models \phi$. Considere una estructura sintáctica de árbol $\mathfrak{A} = (r)$, de manera que $r = \{\boxed{m}\psi, E\}$. Las fórmulas de r se encuentran en el conjunto lean de ϕ y en ese nodo se mantiene $r \Vdash \phi$.
  - Existe una estructura $\mathfrak{M}_t = (W, \mathcal{R}, V)$ que sostiene ϕ en $w \in W$. Para todos los nodos h_i con $(w, h_i) \in R_m$ se sostiene ψ. Aplicando la hipótesis de inducción, existen i estructuras sintácticas de árbol $\mathfrak{A}'_i = (r'_i, \mathfrak{A}'_1, \ldots, \mathfrak{A}'_j)$ y en el nodo r'_i se cumple $\mathfrak{A}'_j \Vdash \psi$. Se crea el nodo $raíz = \{\diamondsuit_m \top, E\}$. A cada nodo r'_i se le agregan las fórmulas $\diamondsuit_m \top$ y $\diamondsuit_m E$. Se establece la estructura sintáctica de árbol $\mathfrak{A} = (raíz, \mathfrak{A}'_1, \ldots, \mathfrak{A}'_j)$ en donde se sostiene $raíz \Vdash \phi$.

Teorema 4. (Completitud). Si una fórmula ϕ se satisface bajo alguna estructura de Kripke $\mathfrak{M} = (W, \mathcal{R}, V)$ en un nodo $r \in W$, entonces el algoritmo devuelve verdadero y una estructura sintáctica de árbol $\mathfrak{A} = (raíz, \mathfrak{A}_1, \ldots, \mathfrak{A}_i)$, considerando la fórmula ϕ como entrada. Por lo que se sostiene que $\mathfrak{M}, r \models \phi$ implica $raíz \Vdash \phi$.

Prueba. La estructura de Kripke $\mathfrak{M}$ se puede convertir en una estructura que posee la propiedad de modelo de árbol $\mathfrak{M}^t$ empleando el Teorema 1, donde ϕ se satisface en el mismo nodo r. La prueba de completitud se realiza mediante inducción sobre la altura de $\mathfrak{M}^t$. Asi mismo, esta prueba es constructiva, porque se especifica como generar la estructura de árbol sintáctico de manera explícita a partir de la estructura de Kripke que satisface la fórmula.

Caso base

- La estructura $\mathfrak{M}^t = (W, \mathcal{R}, V)$ está compuesta por un único nodo: $r \in W$ y en dicho nodo se satisface ϕ. Empleando el Teorema 3, se genera una estructura sintáctica de árbol $\mathfrak{A} = (raíz)$ de manera que se sostiene $raíz \Vdash \phi$.

Paso inductivo

- Se tiene una estructura con forma de árbol $\mathfrak{M}^t = (W, \mathcal{R}, V)$ que satisface ϕ en $r \in W$. Existen subestructuras con forma de árbol $\mathfrak{M}^t_i = (W_i, R_i, V_i)$, tal que su nodo raíz está relacionado con r, es decir, existen las relaciones: $(r, r_i) \in R_m$ y $(r_i, r) \in R_{\overline{m}}$ para alguna modalidad m. Por hipótesis de inducción, el algoritmo

genera estructuras sintácticas de árbol $\mathfrak{A}_i$, con nodos raíces $raíz_{\mathfrak{A}_i}$, por cada estructura $\mathfrak{M}_i^t$.

Empleando el Teorema 3, la estructura $\mathfrak{M}^t$ se puede transformar en una estructura sintáctica de árbol $\mathfrak{A}' = (raíz_{\mathfrak{A}'}, \mathfrak{A}_1, \ldots, \mathfrak{A}_n)$. Se toma el nodo raíz de $\mathfrak{A}'$ y se genera un conjunto $P = \{p_1, \ldots, p_m\}$ de manera que P contiene todas las variables proposicionales de $raíz_{\mathfrak{A}'}$. Posteriormente, considere cada fórmula modal de la forma $\diamondsuit_{\boxed{m}}\varphi$ en el conjunto de fórmulas de $raíz_{\mathfrak{A}'}$, existe algún nodo $raíz_{\mathfrak{A}_i}$ de la estructura sintáctica de árbol $\mathfrak{A}_i$, donde $raíz_{\mathfrak{A}_i} \Vdash \varphi$, en dicho nodo $raíz_{\mathfrak{A}_i}$ se agregan las fórmulas $\diamondsuit_{\boxed{m}}\top$ y $\diamondsuit_{\boxed{m}}(p_1, \ldots, p_m)$, si no están presentes.

Por otro lado, por cada fórmula modal de la forma $\boxed{m}\varphi'$, en el conjunto de fórmulas del nodo $raíz_{\mathfrak{A}'}$ se efectúa lo siguiente: en todos los nodos raíces $raíz_{\mathfrak{A}_i}$ de los arboles sintácticos $\mathfrak{A}_i$ se cumple $raíz_{\mathfrak{A}_i} \Vdash \varphi'$.

Finalmente, si el nodo $raíz_{\mathfrak{A}'}$ sirve como entrada a la función `siguiente` del algoritmo este es capaz de generar todos los nodos raíces de las estructuras sintácticas de árbol $\mathfrak{A}_i$ y devolver una estructura sintáctica de árbol $\mathfrak{A} = (raíz, \mathfrak{A}_1, \ldots, \mathfrak{A}_n)$ donde se cumple $raíz \Vdash \phi$. De esta manera, queda demostrado el Teorema 4.

Capítulo 4

Implementación

Para la implementación del algoritmo de lógica multimodal K se debe determinar el lenguaje de programación apropiado, principalmente que ofrezca un buen rendimiento en cuanto al tiempo de ejecución y que permita manipular la memoria a bajo nivel. Para este proceso de selección se consideraron los lenguajes C, C++, Java, Python, Go y Ocaml.

4.1. Análisis de los lenguajes de programación

El lenguaje Python posee una comunidad muy activa, lo que implica que existe una amplia variedad de librerías que podrían ser útiles para la implementación del algoritmo. Sin embargo, es un lenguaje sumamente lento debido a que el código debe interpretarse en tiempo de ejecución y no se dispone de un manejo de memoria apropiado, especialmente en estructuras de datos dinámicas.

Por otro lado, el lenguaje Java ofrece un rendimiento superior a Python, ya que traduce el código fuente a un código intermedio, previo al lenguaje máquina. Sin embargo, los objetos son estructuras de datos más complejas que podrían impactar en el uso de memoria, en contraste con las estructuras tradicionales como las que existen en C.

En el caso de C y C++, a pesar de que permiten un manejo de memoria preciso y su rendimiento es eficiente, la implementación podría llevar más tiempo en realizarse, debido a que son lenguajes de bajo nivel. No obstante, es probable que el tiempo de ejecución sea considerablemente mejor que en otros lenguajes. Adicionalmente, ambos lenguajes son compilados, por lo que las instrucciones se encuentran en código máquina y su ejecución es directa.

La propuesta final consistió en emplear el lenguaje Go, ya que tiene un rendimiento similar al lenguaje C, en algunas situaciones es aún más rápido en tiempo de ejecución comparado con C, por ejemplo, para el procesamiento de estructuras de datos muy grandes en memoria [8]. Además, puede llegar a ser más rápido en tiempo de ejecución para al-

gunas situaciones comparado Java, por ejemplo, para realizar operaciones con matrices [3].

El lenguaje Go está fortalecido para la programación concurrente, lo que permite a futuro emplear cómputo en la nube para optimizar el algoritmo. Además, su sintaxis está inspirada en lenguajes como Python y C, así la curva de aprendizaje es rápida y disminuye el tiempo de desarrollo. Fue presentado por Google en 2009 y actualmente se emplea en la infraestructura de sistemas distribuidos como alternativa a C++ y Java.

4.2. Implementación del algoritmo

Para que el algoritmo pueda leer las fórmulas y manipularlas se debe usar alguna biblioteca auxiliar. En este trabajo se empleó la biblioteca vartan [37], la cual genera un analizador léxico y sintáctico de tipo LALR(1) necesario para obtener el árbol sintáctico de las fórmulas de entrada. La información que requiere es una gramática libre de contexto en notación Backus-Naur, la especificación de la prioridad de los operadores, su asociatividad y la definición de los símbolos del lenguaje.

En la Figura 4.1 se muestra un fragmento del código que define la gramática libre de contexto para la lógica multimodal K con inversas, así como la prioridad de los operadores codificada para funcionar en vartan. Se observa en el apartado `#prec` los operadores escritos en orden de mayor a menor prioridad. Además se agrega la etiqueta `#left` o `#right` para establecer la asociatividad por la izquierda o derecha, respectivamente.

Posteriormente, se escribe la gramática sin importar el orden. Las variables empleadas para hacerlo corresponden a la definición de símbolos o secuencias de caracteres, que se pueden representar con una expresión regular. Los operadores multimodales `m` y `mi`, están representados por una expresión regular que contiene cadenas de la forma `[1]` y `<2>`, se representan con números naturales entre corchetes o símbolos de "menor y mayor que". Las modalidades inversas son análogas, sin embargo, se agrega una comilla simple seguida del número para representarla, por ejemplo: `<1'>` y `[3']`.

Para definir las variables proposicionales se especificó una expresión regular que contiene cadenas con cualquier letra minúscula del alfabeto (excepto ñ) seguido de un guión bajo y un número natural. Algunos ejemplos de variables proposicionales son: p, q_1, r_{15}. El propósito de incluir el guión bajo y un número es con el objetivo de permitir una fórmula de entrada grande, que no esté limitada por el número de letras del alfabeto.

Adicionalmente, se incluyó la variable proposicional comodín E, que no se debe usar en la fórmula de entrada, no obstante ayuda a clasificarla como tal a nivel sintáctico. Los operadores booleanos se definen con los siguientes caracteres: $\sim$, `/\`, `\/`, `->`, `<->`. Un ejemplo de una fórmula sintácticamente válida es: `(p\/q)/\(r/\<2'>s)/\[1]<3>q`.

```
1
2           #name formula;
3
4           #prec (
5               #left open_parenthesis close_parenthesis
6               #right neg
7               #right m mi
8               #left land
9               #left lor
10              #left imp
11              #left dimp
12          );
13
14          formula
15              : var_prop
16              | tau
17              | contr
18              | neg formula
19              | m formula
20              | mi formula
21              | formula lor formula
22              | formula land formula
23              | formula imp formula
24              | formula dimp formula
25              | open_parenthesis formula close_parenthesis
26              ;
27
```

Figura 4.1: Gramática de la lógica multimodal K con inversas codificada en la biblioteca vartan en notación Backus-Naur.

La biblioteca vartan posee una herramienta para compilar la gramática y generar tres archivos de código fuente listos para utilizar en la implementación del algoritmo. Entre estos archivos se encuentra una función que permite obtener el árbol sintáctico dada una cadena de caracteres, que en este caso corresponde con la fórmula de entrada de lógica multimodal K con inversas.

Posteriormente, a partir del árbol sintáctico, se realiza la conversión de la fórmula a la forma normal negativa y se envía el árbol sintáctico al algoritmo de satisfacción. Después se genera el conjunto *lean*. El conjunto *lean* es una lista de cadenas de caracteres que contiene la información significativa de la fórmula. Todos los nodos únicamente pueden tener fórmulas de este conjunto y algunas adicionales que se detallan en la Definición 8 (nodos), por lo que en la implementación cuando se requiere almacenar una fórmula en

un nodo se emplea un apuntador a algún elemento de la lista del *lean*.

En su mayor parte se trató de reutilizar la memoria del programa mediante el uso de apuntadores, los cuales generalmente tienen un tamaño de 8 bytes en sistemas operativos de 64 bits. Por ejemplo, si cada nodo almacena una lista de fórmulas las cuales son apuntadores a cadenas de caracteres y si se almacenan 100 fórmulas, la lista ocuparía un total de 800 bytes. En cambio, si se considera una lista de fórmulas que almacena cadenas de caracteres, si se almacena el mismo número de fórmulas, suponiendo que el promedio de la longitud de las fórmulas es de 15, la lista consumiría un total de 1500 bytes.

Lo anterior es únicamente un ejemplo de como el uso de apuntadores es beneficioso para el algoritmo, ya que con fórmulas mucho más grandes, la cantidad de nodos se va incrementando, así como el tamaño de las fórmulas. Adicionalmente, como los nodos almacenan algunas fórmulas modales que se satisfacen en los nodos ancestros, a mayor profundidad modal, aumenta el número y tamaño de estas fórmulas.

El algoritmo después de generar los nodos hijos en la función `siguiente`, debe comprobar que los nodos no se repitan en el árbol. Debido a esto se debe hacer una comparación uno a uno entre cada nodo generado y todos los nodos actuales del árbol. Para hacer más eficiente verificar que dos nodos son o no son iguales, en primera instancia se considera el tamaño de la lista de apuntadores a fórmulas del conjunto *lean* que poseen. Si no coincide, entonces ya no es necesario verificar, puesto que se puede concluir que no son iguales.

En caso contrario, si el tamaño de ambas listas de fórmulas coincide, se realiza una comparación de apuntadores, lo que disminuye el tiempo de búsqueda porque se evita comparar cadenas de caracteres, las cuales pueden llegar a ser demasiado grandes. Esta optimización en la implementación es necesaria porque la operación se realiza en cada llamada a la función `siguiente`, la cual se llama una vez por cada nodo hijo generado que ingresa a la pila de nodos.

Cuando el algoritmo está generando la estructura sintáctica de árbol y falla la creación de alguna rama, ya sea que algún nodo hijo generado se repita o que no cumpla las propiedades para ser un nodo, por ejemplo, que se haya encontrado contradicciones con respecto a las variables proposicionales, entonces se realiza la operación de *backtracking*.

Generalmente el *backtracking* se realiza empleando funciones recursivas, las cuales facilitan la implementación. No obstante, la recursividad en nuestro caso no es eficiente, puesto que por cada llamada de la función se generan copias de las variables auxiliares que se emplean en el cuerpo de la función recursiva. Por otro lado, el flujo de la ejecución es difícil de rastrear cuando se desea hacer una depuración del programa.

Por lo tanto, en este algoritmo se optó por realizar un flujo lineal mediante ciclos iterativos. Las operaciones de *backtracking* se simulan empleando una pila auxiliar que va

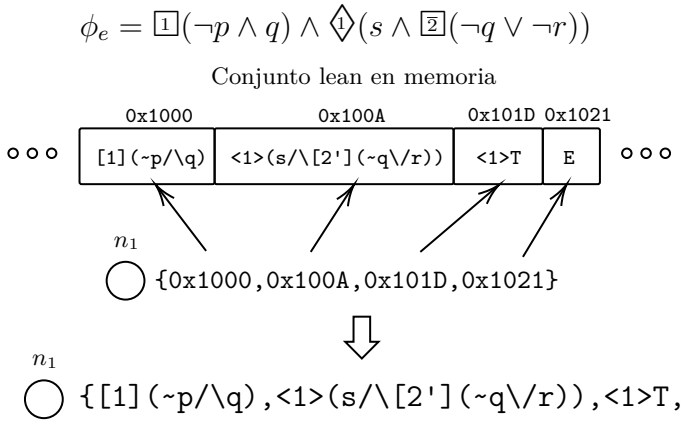


Figura 4.2: Ejemplo de representación del *lean* y nodos en memoria.

almacenando los nodos del árbol a los cuales se debe regresar cuando alguna de las ramas falla.

4.3. Optimizaciones

A manera de resumen, en esta subsección se enlistaran las optimizaciones consideradas para la implementación del algoritmo y algunos ejemplos.

Uso de memoria dinámica. Una vez que se construye el conjunto *lean*, este se almacena en memoria y se reutiliza para la generación de los nodos. Por lo que cada nodo es una lista de apuntadores al *lean*. En la Figura 4.2 se muestra de manera gráfica un ejemplo de como las fórmulas del nodo w_1 se pueden codificar con base en las direcciones de memoria donde están almacenadas las fórmulas del *lean*.

Funcionamiento iterativo. El algoritmo se implementó con un enfoque iterativo, por lo que la recursión se evita con el objetivo es disminuir el consumo de memoria, que es originado por las variables locales de la función que se llama. Adicionalmente los llamados de la función se acumulan en la pila de llamadas. Una mejor práctica consiste en emplear una pila para acumular los parámetros de la función y utilizar un ciclo que los extrae y procesa uno a uno, hasta que la pila se vacía.

Fusión de nodos diamante. Cuando sea posible el algoritmo intentará realizar una fusión de nodos. Por ejemplo, si se tiene la fórmula $\phi_f = \langle\!\langle 1 \rangle\!\rangle p \wedge \langle\!\langle 1 \rangle\!\rangle q$ se puede observar que se debe satisfacer p y q en algún mundo de una estructura que se relaciona con el nodo raíz mediante la modalidad 1. En este caso, es posible construir una estructura sintáctica de árbol que posee un único nodo hijo que contiene ambas fórmulas, ya que esto no ocasiona contradicciones. En la Figura 4.3 se muestra de manera gráfica la estructura sintáctica de árbol $\mathfrak{A}$ que considera este caso.

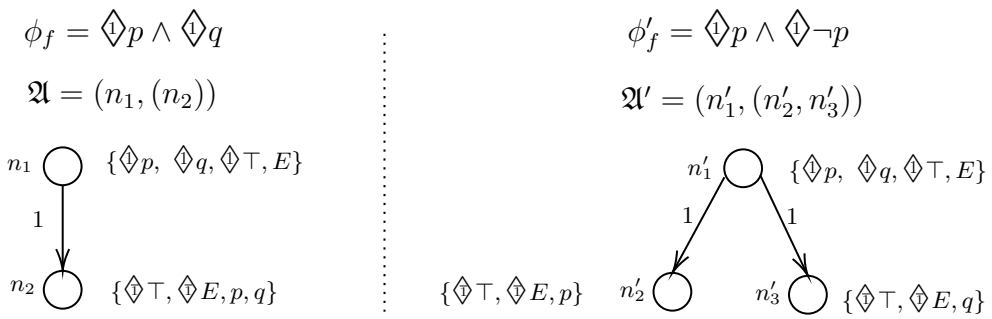


Figura 4.3: Ejemplo de dos estructuras sintácticas de árbol, tal que se cumple $n_1 \Vdash \phi_f$ y $n'_1 \Vdash \phi'_f$.

No obstante, si por alguna contradicción las variables no pudieran estar contenidas en un mismo nodo, como sucede en el caso de tener la fórmula $\phi'_f = \Diamond_1 p \wedge \Diamond_1 \neg p$, los nodos terminan construyéndose por separado, ya que considerar p y $\neg p$ en un mismo nodo provoca una contradicción. El resultado se observa en la estructura sintáctica $\mathfrak{A}'$ de la misma figura.

Cabe resaltar que esto solo aplica a fórmulas modales con operador $\Diamond$ y que posean las mismas modalidades. Además, el algoritmo si encuentra algún tipo de contradicción futura, es decir, en un nodo más profundo ocasionada por esta fusión, llegará al caso de intentar construir la estructura separando los nodos.

Capítulo 5

Experimentación

Para la implementación del algoritmo se empleó la última versión disponible del lenguaje Go, la versión 1.20 y el desarrollo continuó hasta la versión 1.22, donde compila exitosamente. Para medir el rendimiento del algoritmo, se utilizaron dos bibliotecas llamadas `testing` y `benchstat`, la primera forma parte de la biblioteca estándar de Go y la segunda debe instalarse por separado.

La medición se realiza sobre la función principal del programa, que incluye la lectura de la fórmula, la generación de su árbol sintáctico, la conversión a la forma normal negativa, así como el proceso algorítmico de determinar si la fórmula se satisface o no. No se tiene en cuenta el tiempo que tarda en cargar el ejecutable y las bibliotecas en memoria.

Con el objetivo de medir empíricamente el rendimiento del algoritmo y tener una referencia cuantificable, se generaron manualmente algunas fórmulas que se satisfacen bajo algún mundo de una estructura de Kripke y fórmulas que no se satisfacen bajo ninguna estructura, además se realizó experimentación con algunas fórmulas obtenidas de trabajos relacionados, de [27] y de un conjunto de fórmulas conocido como TANCS2000 [31].

En el caso del conjunto de fórmulas creadas manualmente, se define cada una a nivel de implementación como un elemento de una variable de tipo `slice` (muy similar a una lista). Posteriormente se define una función, cuyo nombre debe comenzar necesariamente con el prefijo `Benchmark` debido a las convenciones del lenguaje. Con ayuda del comando `go test` se llama la función anterior, la cual ejecuta el algoritmo tomando cada fórmula de la lista. Adicionalmente, en dicho comando se definen algunos parámetros de configuración para indicar que se realicen $1,000$ llamadas a la función principal del algoritmo, ya que internamente calcula la media de estas llamadas, y que este proceso se repita 10 veces.

Para el conjunto de fórmulas pertenecientes al TANCS2000, se crearon archivos de texto de manera que cada uno almacena una fórmula. Posteriormente se ejecuta una función que sigue la misma nomenclatura que la mencionada en el párrafo anterior, que lee cada archivo y después llama al algoritmo. Como estas fórmulas son mucho más complejas se

estableció una única llamada y el proceso se repitió 6 veces, que es lo mínimo para tener un intervalo de confianza por encima del 95 %. Las configuraciones para obtener buenas mediciones se obtuvieron de manera experimental.

Posteriormente, se utiliza el comando `benchstat` para generar un resumen de los resultados. Por cada fórmula se calcula el tiempo medio y el intervalo de confianza. El intervalo de confianza considera la dispersión que existe en las mediciones realizadas, por lo que el caso ideal es obtener un valor pequeño, lo que significa que las mediciones de rendimiento son fiables.

Los experimentos se realizaron en una computadora con un procesador Intel Core i5-4300U a 1,90GHz con una velocidad máxima de 2,49GHz, 12GB de RAM DDR3 en sistema operativo Windows 10. Con el objetivo de disminuir la variabilidad en las mediciones se consideró una instalación fresca del sistema operativo y sin ningún tipo de anti-virus.

5.1. Fórmulas que se satisfacen bajo alguna estructura

Para esta sección, se realizaron dos experimentos distintos. El primero de estos, como se muestra en la Tabla 5.1, consiste en un conjunto de fórmulas de diferentes profundidades modales máximas y con un número de a lo más de 5 variables proposicionales.

Los tiempos presentados resultan satisfactorios debido a que el tiempo de ejecución está alrededor de los milisegundos. Se puede observar que la profundidad modal máxima puede influir en el tiempo de ejecución, pero por supuesto hay distintos factores. Uno de ellos es la anchura de la fórmula, ya que eso provocará que la estructura sintáctica de árbol resultante posea múltiples subárboles.

Un segundo factor que influye en el tiempo de ejecución, es la cantidad de posibles árboles que prueba el algoritmo antes de llegar a una conclusión de que la fórmula se satisface. Por ejemplo, una fórmula de este tipo puede contener contradicciones en las últimas ramas, en un nivel profundo, por lo que el algoritmo debe realizar una vuelta atrás, que en el peor caso, es necesario probar con un nodo raíz distinto. Además, es posible que este proceso se repita una cantidad de veces considerable, dependiendo de la fórmula de entrada.

Como segundo experimento, para verificar el impacto en el algoritmo de la anchura y profundidad de los árboles construidos, se generaron fórmulas modales con distintas configuraciones, las cuales solamente poseen variables proposicionales en los nodos hojas. Este experimento se pueden observar en la Tabla 5.2.

Se puede apreciar que el tiempo de ejecución incrementa si la profundidad modal se mantiene fija y la anchura variable, como era de esperarse. Lo mismo sucede si la anchura se

Tabla 5.1: Tiempo de ejecución de fórmulas que se satisfacen bajo algún mundo de alguna estructura de Kripke.

ID	Formula	Profundidad modal máx.	Tiempo de ejecución (ms)
$S1$	$\neg p \wedge r \wedge \neg(p \wedge \neg q \vee s) \vee p \wedge q$	0	$0.106 \pm 2\,\%$
$S2$	$(\neg p \wedge \neg q \wedge r \wedge \neg(r \leftrightarrow q)) \wedge \neg(\boxed{2}(p \vee r) \wedge \Diamond_{2}\neg p)$	1	$0.270 \pm 2\,\%$
$S3$	$\neg(\boxed{1}(\neg p \wedge \neg q)) \wedge \Diamond_{1}\neg(\Diamond_{1} p \leftrightarrow \Diamond_{1} q) \wedge (p \wedge \neg p \vee q \vee r \wedge s)$	2	$0.623 \pm 0\,\%$
$S4$	$\Diamond_{1}((p \wedge \Diamond_{2}(q \rightarrow r)) \vee \Diamond_{2}(q \vee r) \vee s) \wedge \Diamond_{1}((p \wedge \Diamond_{1} q \wedge \boxed{2}(q \wedge \neg r)) \vee \Diamond_{1}\Diamond_{2}(p \wedge \neg p) \vee \Diamond_{2}(\neg q \wedge \neg r))$	3	$1.290 \pm 0\,\%$
$S5$	$\Diamond_{\bar{1}}(\Diamond_{2}\Diamond_{3}(p \wedge q) \wedge \boxed{2}(r \rightarrow s)) \wedge \Diamond_{\bar{1}}(\Diamond_{2}(r \rightarrow s) \wedge (\boxed{2}(\boxed{3}(\neg p \wedge \neg q) \vee \Diamond_{3}(\neg p \wedge \neg q))))$	3	$2.108 \pm 2\,\%$
$S6$	$\Diamond_{1}\Diamond_{2}\Diamond_{3}(p \rightarrow q) \wedge \Diamond_{1}(\boxed{2}(p \wedge (\boxed{3}(p \wedge \neg q) \vee \boxed{3}(p \wedge q) \vee s))) \wedge (\Diamond_{1}\Diamond_{2}\boxed{3}(p \wedge q) \vee \Diamond_{1}\Diamond_{2}\boxed{3}(\neg p \wedge \neg q) \ \wedge \neg s)$	3	$2.859 \pm 1\,\%$
$S7$	$\Diamond_{3}((p \wedge \neg p) \vee \Diamond_{2}(t \wedge \Diamond_{2}(s \wedge \Diamond_{\bar{1}}(p \rightarrow q \vee r)))) \wedge \boxed{3}p \wedge \neg\Diamond_{3}\Diamond_{2}(t \wedge \neg p)$	4	$0.683 \pm 1\,\%$
$S8$	$\neg((p \leftrightarrow s) \vee \boxed{1}(p \ \wedge \neg r \ \wedge \neg q \vee \boxed{2}(q \vee \boxed{\bar{1}}(p \vee \Diamond_{3} s))) \vee \boxed{2}\neg q) \wedge \Diamond_{1}\Diamond_{2}\Diamond_{\bar{1}}(\neg s \wedge \neg\boxed{3}(r \wedge p \vee q))$	4	$1.533 \pm 1\,\%$
$S9$	$\Diamond_{\bar{1}}(\boxed{2}(p \wedge \neg p) \vee \Diamond_{2} q) \wedge \Diamond_{\bar{1}}(p \wedge \Diamond_{2}\Diamond_{1}\Diamond_{3}(p \wedge \Diamond_{2} q)) \wedge \boxed{\bar{1}}(\neg p \vee q)$	5	$0.923 \pm 0\,\%$
$S10$	$\neg\Diamond_{1}(p \leftrightarrow \Diamond_{2}(q \vee r)) \wedge \Diamond_{1}\boxed{\bar{1}}\boxed{1}p \wedge \Diamond_{2}(\Diamond_{4} t \wedge \Diamond_{3}\Diamond_{\bar{1}}\Diamond_{\bar{1}}\neg(\boxed{\bar{1}}\Diamond_{2}\neg q \vee \boxed{\bar{1}}\boxed{2}r)) \wedge \Diamond_{2}(\Diamond_{3}\Diamond_{4} s)$	6	$3.412 \pm 5\,\%$

mantiene fija y la profundidad varia. No obstante algo interesante a recalcar es que los tiempos de ejecución aumentan en mayor proporción cuando la fórmula es mucho más profunda.

Esto se debe a que cada nodo de la estructura sintáctica de árbol que genera el algoritmo contiene formulas especiales que se cumplen en los nodos ancestros. Estas fórmulas contienen una conjunción de las variables proposicionales de dichos nodos. Por esta razón, un nodo en el octavo nivel de la estructura sintáctica de árbol tendrá al menos siete fórmulas adicionales, inherentes a la forma del árbol.

El costo de generar estas fórmulas comienza a impactar si tanto la profundidad como la anchura incrementan, cuando el árbol comienza a tener una dimensión considerable, este tipo de fórmulas tienen un costo sobre el algoritmo.

Tabla 5.2: Tiempo de ejecución de fórmulas que se satisfacen bajo algún mundo de alguna estructura de Kripke, considerando que únicamente se cumplen variables proposicionales en la profundidad modal máxima.

Formula	Anchura	Tiempo de ejecución (ms)
$(p \wedge \neg p) \vee (q \rightarrow \neg p)$	1	$0.050 \pm 4\,\%$
Profundidad modal 1		
$\Diamond_1((p \wedge \neg p) \vee (q \rightarrow \neg p))$	1	$0.099 \pm 1\,\%$
$\Diamond_1((p \wedge \neg p) \vee (q \rightarrow \neg p)) \wedge \Diamond_2((p \wedge \neg p) \vee (q \rightarrow \neg p))$	2	$0.190 \pm 3\,\%$
...	4	$0.374 \pm 5\,\%$
...	8	$0.730 \pm 2\,\%$
...	16	$1.196 \pm 1\,\%$
Profundidad modal 2		
$\Diamond_1(\Diamond_1(p \wedge \neg p) \vee \Diamond_1(q \rightarrow \neg p))$	1	$0.162 \pm 1\,\%$
$\Diamond_1(\Diamond_1(p \wedge \neg p) \vee \Diamond_1(q \rightarrow \neg p)) \wedge \Diamond_2(\Diamond_1(p \wedge \neg p) \vee \Diamond_1(q \rightarrow \neg p))$	2	$0.320 \pm 2\,\%$
...	4	$0.643 \pm 4\,\%$
...	8	$1.248 \pm 1\,\%$
...	16	$2.635 \pm 1\,\%$
Profundidad modal 4		
$\Diamond_1(\Diamond_1\Diamond_2\Diamond_3(p \wedge \neg p) \vee \Diamond_1\Diamond_2\Diamond_3(q \rightarrow \neg p))$	1	$0.448 \pm 5\,\%$
$\Diamond_1(\Diamond_1\Diamond_2\Diamond_3(p \wedge \neg p) \vee \Diamond_1\Diamond_2\Diamond_3(q \rightarrow \neg p)) \wedge \Diamond_2(\Diamond_1\Diamond_2\Diamond_3(p \wedge \neg p) \vee \Diamond_1\Diamond_2\Diamond_3(q \rightarrow \neg p))$	2	$0.883 \pm 5\,\%$
...	4	$2.162 \pm 6\,\%$
...	8	$6.038 \pm 4\,\%$
...	16	$31.51 \pm 5\,\%$
Profundidad modal 8		
$\Diamond_1(\Diamond_1\Diamond_2\Diamond_3\Diamond_4\Diamond_5\Diamond_6\Diamond_7(p \wedge \neg p) \vee \Diamond_1\Diamond_2\Diamond_3\Diamond_4\Diamond_5\Diamond_6\Diamond_7(q \rightarrow \neg p))$	1	$1.535 \pm 6\,\%$
$\Diamond_1(\Diamond_1\Diamond_2\Diamond_3\Diamond_4\Diamond_5\Diamond_6\Diamond_7(p \wedge \neg p) \vee \Diamond_1\Diamond_2\Diamond_3\Diamond_4\Diamond_5\Diamond_6\Diamond_7(q \rightarrow \neg p)) \wedge \Diamond_2(\Diamond_1\Diamond_2\Diamond_3\Diamond_4\Diamond_5\Diamond_6\Diamond_7(p \wedge \neg p) \vee \Diamond_1\Diamond_2\Diamond_3\Diamond_4\Diamond_5\Diamond_6\Diamond_7(q \rightarrow \neg p))$	2	$5.101 \pm 1\,\%$
...	4	$31.47 \pm 2\,\%$
...	8	$311.3 \pm 3\,\%$
...	16	$3485 \pm 1\,\%$

5.2. Fórmulas que no se satisfacen bajo ninguna estructura

En este tipo de fórmulas, se realizó una experimentación que se muestra en la Tabla 5.3. Se pueden apreciar fórmulas de distintas profundidades modales máximas y a lo más ocho variables proposicionales.

Tabla 5.3: Tiempo de ejecución de fórmulas que no se satisfacen bajo ninguna estructura de Kripke.

Id.	Formula	Profundidad modal max.	Tiempo de ejecución (ms)
$I1$	$\Diamond_{\overline{1}} p \wedge \Box_{\overline{1}}(p \rightarrow q) \wedge \Diamond_{\overline{1}}(\neg q \wedge p)$	1	$0.114 \pm 1\,\%$
$I2$	$\Diamond_1((p \rightarrow q) \wedge \Diamond_2(r \vee s)) \wedge \Box_1((p \wedge \neg p) \vee \neg(q \rightarrow \neg p)) \wedge \Diamond_2(\neg r \wedge \neg s) \wedge \Diamond_1 \neg p$	2	$0.278 \pm 1\,\%$
$I3$	$\neg\Diamond_1(p \wedge q) \wedge \neg\Diamond_1\Diamond_2(p \rightarrow q) \wedge \Diamond_1\Diamond_2(\neg p \wedge \neg q)$	2	$0.355 \pm 1\,\%$
$I4$	$\Diamond_1(\Diamond_2 \neg p \wedge (\Diamond_3(p \leftrightarrow q))) \wedge \Diamond_1\Diamond_2\Box_3(p \wedge \neg q) \wedge \Box_1\Box_2 p$	3	$0.461 \pm 1\,\%$
$I5$	$\Diamond_1\Diamond_{\overline{1}}(\Diamond_1(p \leftrightarrow s) \wedge \Box_{\overline{1}}(p \wedge q)) \wedge \Diamond_1\Diamond_{\overline{1}}\Diamond_1(r \vee s \rightarrow t) \wedge \Box_1(\Diamond_{\overline{1}}(t \vee (w \wedge x) \rightarrow s)) \wedge \Diamond_{\overline{1}}\Box_1\Diamond_{\overline{1}}(r \wedge s \wedge \neg t) \wedge \Diamond_1\Diamond_{\overline{1}}(\neg s \wedge t) \wedge \Box_1\Box_{\overline{1}}\neg t$	3	$3.983 \pm 0\,\%$
$I6$	$\Diamond_1\Box_2((p \rightarrow q) \wedge (r \vee s) \wedge \Diamond_3((t \rightarrow v) \vee w) \wedge \Box_4((\neg r \leftrightarrow x) \wedge \Diamond_5 s)) \wedge \Box_1\Diamond_2((r \rightarrow s) \wedge \Box_3(\neg w \vee (\neg t \wedge v)) \wedge \Diamond_4((\neg r \wedge \neg x) \wedge \neg\Diamond_5 s))$	4	$1.793 \pm 0\,\%$
$I7$	$S1 \wedge S2 \wedge S3 \wedge (I1 \vee I2 \vee I3)$	2	$6.569 \pm 2\,\%$
$I8$	$S1 \wedge S3 \wedge S5 \wedge S6 \wedge (I1 \vee I2 \vee I3 \vee I4)$	3	$2,524 \pm 1\,\%$
$I9$	$S10 \wedge I6$	6	$8.016 \pm 7\,\%$
$I10$	$S10 \wedge S9 \wedge I6$	6	$19.69 \pm 1\,\%$

Los tiempos de ejecución para estas fórmulas son razonables. Al compararlas con las fórmulas que se satisfacen de la sección anterior, aparentemente son tiempos muy similares. Esto se debe a que, intuitivamente, se podría pensar que para este tipo de fórmulas el algoritmo debe probar todas las posibles estructuras sintácticas de árbol. Lo cual en el peor caso es cierto.

No obstante, puede haber casos en los que la contradicción se encuentra en un nivel superior del árbol e inclusive en la primera rama que el algoritmo intenta construir. En estos casos no es necesario finalizar la construcción del árbol y puede suceder que únicamente se genere el nodo raíz, lo que puede ocasionar que el tiempo de ejecución disminuya.

En los resultados presentados, la contradicción se puede encontrar en cualquier parte de la estructura sintáctica de árbol, por lo que los tiempos de ejecución pueden ser mejores inclusive si la fórmula es mucho más grande y con mayor profundidad modal. Esto sucede en la fórmula con Id. $I6$, en contraste con la que posee Id. $I5$.

Tabla 5.4: Comparación con respecto al tiempo de ejecución reportado en [27] y el algoritmo presentado en este trabajo.

Formula	Profundidad modal max.	Tiempo de ejecución (ms) en [27]	**Tiempo de ejecución (ms)**
$\langle 1 \rangle a$	1	2153	$0.084 \pm 3\,\%$
$\langle 1 \rangle a \wedge b$	1	5180	$0.090 \pm 1\,\%$
$\langle 1 \rangle a \wedge \langle 2 \rangle b$	1	11419	$0.147 \pm 1\,\%$
$\langle 1 \rangle a \wedge \langle i \rangle b$	1	9188	$0.147 \pm 1\,\%$
$\langle 1 \rangle a \wedge \langle 2 \rangle b$	1	9110	$0.148 \pm 3\,\%$
$\langle 1 \rangle a \wedge \langle 2 \rangle b \wedge c$	1	42604	$0.153 \pm 1\,\%$
$\langle 1 \rangle a \wedge \langle 2 \rangle b \wedge \neg c$	1	41403	$0.155 \pm 2\,\%$
$\langle 1 \rangle a \wedge \langle 2 \rangle b \wedge \neg(c \vee d)$	1	38937	$0.202 \pm 2\,\%$
$\langle 1 \rangle \langle 1 \rangle a$	2	157359	$0.153 \pm 1\,\%$
$\langle 1 \rangle \langle 1 \rangle a \wedge b$	2	318714	$0.160 \pm 1\,\%$
$\langle 1 \rangle \langle 1 \rangle a \wedge \langle 1 \rangle b$	2	863249	$0.219 \pm 1\,\%$
$\langle 1 \rangle \langle 1 \rangle a \wedge \langle i \rangle b$	2	595539	$0.223 \pm 1\,\%$
$\langle 1 \rangle \langle 1 \rangle a \wedge \langle 2 \rangle b$	2	897349	$0.223 \pm 1\,\%$
$\langle 1 \rangle \langle 1 \rangle a \wedge \langle 2 \rangle b$	2	904157	$0.223 \pm 2\,\%$
$\langle 1 \rangle \langle 1 \rangle a \wedge \langle 2 \rangle \langle 2 \rangle b$	2	2043845	$0.293 \pm 1\,\%$

Las fórmulas con Id. $I7 - I10$ son de tamaño mucho mayor comparadas con las demás, debido a que para construirlas se emplean conjunciones de fórmulas que se satisfacen y disyunciones de fórmulas que no se satisfacen. A pesar de que no es una técnica eficiente, debido a que la conjunción de dos fórmulas que se satisfacen puede originar alguna contradicción y finalizar la construcción de la estructura sintáctica de árbol mucho antes, se logró generar un caso en el que el tiempo de ejecución aumenta al orden de segundos, como se muestra en la fórmula con Id. $I8$.

5.3. Fórmulas de trabajos relacionados

En esta sección se presentan los experimentos con fórmulas extraídas de artículos relacionados. Las fórmulas del primer trabajo [27] pertenecen a la experimentación realizada con un algoritmo de satisfacción para lógica modal K, que emplea el mismo enfoque que nuestro algoritmo, ya que se basa en la propiedad de modelo de árbol. Los resultados se muestran en la Tabla 5.4. Se pueden visualizar fórmulas de hasta cuatro variables proposicionales y con una profundidad modal máxima de dos, que se satisfacen bajo alguna

Tabla 5.5: Conjunto de fórmulas: Easy TANCS2000 - Parte 1.

No.	C	V	D	T. de E. (seg.)	No.	C	V	D	T. de E. (seg.)
1	20	8	4	$19.49 \pm 2\,\%$	3	20	8	6	$31.16 \pm 4\,\%$
2	20	8	4	$38.45 \pm 4\,\%$	4	20	8	6	$184.0 \pm 2\,\%$
3	20	8	4	$38.22 \pm 1\,\%$	5	20	8	6	$151.0 \pm 1\,\%$
4	20	8	4	$22.94 \pm 2\,\%$	6	20	8	6	$36.73 \pm 4\,\%$
5	20	8	4	$23.19 \pm 4\,\%$	7	20	8	6	$16.59 \pm 3\,\%$
6	20	8	4	$14.53 \pm 2\,\%$	8	20	8	6	$351.2 \pm 1\,\%$
7	20	8	4	$29.84 \pm 2\,\%$	1	30	16	4	$3.098k \pm 2\,\%$
8	20	8	4	$6.249 \pm 6\,\%$	2	30	16	4	$1.694k \pm 2\,\%$
1	20	8	6	$1.423 \pm 2\,\%$	3	30	16	4	$2.938k \pm 4\,\%$
2	20	8	6	$54.91 \pm 3\,\%$	4	30	16	4	$2.402k \pm 2\,\%$

estructura.

Si se compara el tiempo de ejecución entre la propuesta de este trabajo y la del artículo en cuestión, se muestra una mejora substancial. Una de las razones y la más evidente, se debe a que en [27] se realiza una construcción exhaustiva de todos los árboles posibles incrementando su complejidad en cada iteración. Por lo que el espacio de búsqueda es demasiado amplio y crece acorde al tamaño de la fórmula de entrada, debido a que el conjunto *lean* es proporcional al tamaño de dicha fórmula.

Por otro lado, dada la fórmula de entrada y el conjunto *lean*, nuestro algoritmo es capaz de identificar los elementos del *lean* para construir nodos válidos desde la raíz y desarrollar las posibles estructuras sintácticas de árbol una a una. Por lo que si no se presentan contradicciones en el primer intento de construcción de la estructura sintáctica de árbol, el algoritmo puede concluir que la fórmula se satisface mucho antes.

Para describir la segunda experimentación, primeramente se detalla un poco acerca del origen del conjunto TANCS2000. En 1998 se llevó a cabo la primera conferencia de nombre *TANCS Non Classical System Comparison*. Era una competición de probadores automáticos de teoremas para lógicas no clásicas, especialmente lógica modal K y lógica temporal KT. En dicha competencia se definieron algunas especificaciones sobre cómo medir el rendimiento de aquellos algoritmos y se generó el conjunto conocido en la literatura como TANCS2000 [31].

El proceso empleado en la generación de las fórmulas de lógica modal K, en el conjunto TANCS2000, consiste en construir fórmulas booleanas cuantificadas (QBF) válidas que, posteriormente, se traducen a fórmulas de lógica modal K de manera que se satisfacen bajo alguna estructura de Kripke. En [31] se describen tres métodos que son utilizados en la traducción de las fórmulas. Este conjunto hace uso del método de traducción descrito en [23].

Algunas características de las fórmulas QBF, generadas como paso intermedio, son el uso de cuantificadores que se van alternando entre sí y la representación en forma normal conjuntiva. La nomenclatura de los archivos es la siguiente: `p-qbf-cnfSSS-K4-C#-V#-D#.#.txt`. Donde el símbolo `#` representa un valor entero positivo y son parámetros del algoritmo que construye las fórmulas. Dichos parámetros personalizan las características de las fórmulas QBF que se generan.

Con base en lo anterior, la variable `C` indica el número de cláusulas en forma normal conjuntiva de la fórmula, `V` señala el número de variables que cuantifican los operadores alternantes y `D` representa la profundidad de alternancia de la fórmula. El último `#` señala el identificador de la fórmula (No.) cuando hay más de una que posee las mismas especificaciones.

En las Tablas 5.5 y 5.6 se muestran los resultados de la ejecución del algoritmo sobre el conjunto llamado *easy*, el cual está compuesto por fórmulas modales de una única modalidad y no contiene modalidades inversas. El significado de cada columna está asociado a lo descrito en el párrafo anterior, adicionando "T. de E." como tiempo de ejecución.

Se puede apreciar que algunas fórmulas son resueltas en el orden de los milisegundos, mientras el valor más grande corresponde a horas. En general, el algoritmo termina en un tiempo razonable, no obstante, la razón por la que no se establece un tiempo límite es con el objetivo de identificar aquellas fórmulas con posibilidad de mejora, además de establecer valores de referencia para trabajos futuros.

El número de cláusulas y el número de variables tienen un impacto significativo en el tiempo de ejecución conforme se incrementa su valor, debido a que dichos parámetros influyen en el tamaño de la fórmula QBF y es posible que también en la resultante. Adicionalmente, la profundidad de alternancia también incrementa la dificultad de la fórmula, no obstante se observa que influye en el tiempo de ejecución a una menor escala.

Tabla 5.6: Conjunto de fórmulas: Easy TANCS2000 - Parte 2.

No.	C	V	D	T. de E. (seg.)	No.	C	V	D	T. de E. (seg.)
1	10	16	4	$242.0 \pm 3\,\%$	3	10	8	6	$32.53 \pm 2\,\%$
2	10	16	4	$67.17 \pm 1\,\%$	4	10	8	6	$20.82 \pm 4\,\%$
3	10	16	4	$39.29 \pm 1\,\%$	5	10	8	6	$1.666 \pm 3\,\%$
4	10	16	4	$76.20 \pm 2\,\%$	6	10	8	6	$26.45 \pm 0\,\%$
7	10	16	4	$84.98 \pm 2\,\%$	7	10	8	6	$33.07 \pm 2\,\%$
8	10	16	4	$74.87 \pm 2\,\%$	8	10	8	6	$32.05 \pm 5\,\%$
1	10	16	6	$1.184k \pm 2\,\%$	1	20	16	4	$53.43 \pm 3\,\%$
2	10	16	6	$429.0 \pm 0\,\%$	2	20	16	4	$865.2 \pm 2\,\%$
3	10	16	6	$494.3 \pm 2\,\%$	3	20	16	4	$1.294k \pm 0\,\%$
4	10	16	6	$98.22 \pm 4\,\%$	4	20	16	4	$246.1 \pm 2\,\%$
5	10	16	6	$587.5 \pm 1\,\%$	5	20	16	4	$1.959k \pm 9\,\%$
6	10	16	6	$323.4 \pm 2\,\%$	6	20	16	4	$1.204k \pm 2\,\%$
7	10	16	6	$19.09 \pm 2\,\%$	7	20	16	4	$730.1 \pm 15\,\%$
8	10	16	6	$423.2 \pm 2\,\%$	8	20	16	4	$1.401k \pm 1\,\%$
1	10	4	4	$80.17m \pm 121\,\%$	1	20	16	6	$2.552k \pm 1\,\%$
2	10	4	4	$223.1m \pm 4\,\%$	2	20	16	6	$5.013k \pm 1\,\%$
3	10	4	4	$136.4m \pm 7\,\%$	5	10	16	4	$99.94 \pm 1\,\%$
4	10	4	4	$27.82m \pm 15\,\%$	6	10	16	4	$21.97 \pm 2\,\%$
5	10	4	4	$10.98m \pm 28\,\%$	3	20	16	6	$6.020k \pm 1\,\%$
6	10	4	4	$162.9m \pm 9\,\%$	4	20	16	6	$7.082k \pm 3\,\%$
7	10	4	4	$369.9m \pm 6\,\%$	5	20	16	6	$5.575k \pm 3\,\%$
8	10	4	4	$25.63m \pm 11\,\%$	6	20	16	6	$4.065k \pm 6\,\%$
1	10	4	6	$207.8m \pm 8\,\%$	7	20	16	6	$499.0 \pm 1\,\%$
2	10	4	6	$985.9m \pm 1\,\%$	8	20	16	6	$2.262k \pm 2\,\%$
3	10	4	6	$32.90m \pm 16\,\%$	1	20	4	4	$710.4m \pm 3\,\%$
4	10	4	6	$889.7m \pm 3\,\%$	2	20	4	4	$984.3m \pm 3\,\%$
5	10	4	6	$495.6m \pm 3\,\%$	3	20	4	4	$1.225 \pm 2\,\%$
6	10	4	6	$357.7m \pm 11\,\%$	4	20	4	4	$955.6m \pm 9\,\%$
7	10	4	6	$1.049 \pm 9\,\%$	5	20	4	4	$494.9m \pm 4\,\%$
8	10	4	6	$840.0m \pm 3\,\%$	6	20	4	4	$1.173 \pm 3\,\%$
1	10	8	4	$1.390 \pm 4\,\%$	7	20	4	4	$743.2m \pm 3\,\%$
2	10	8	4	$3.361 \pm 21\,\%$	8	20	4	4	$63.90m \pm 15\,\%$
3	10	8	4	$3.320 \pm 1\,\%$	1	20	4	6	$6.442 \pm 2\,\%$
4	10	8	4	$1.229 \pm 2\,\%$	2	20	4	6	$7.258 \pm 0\,\%$
5	10	8	4	$1.621 \pm 2\,\%$	3	20	4	6	$850.7m \pm 1\,\%$
6	10	8	4	$2.062 \pm 4\,\%$	4	20	4	6	$5.021 \pm 1\,\%$
7	10	8	4	$167.9m \pm 8\,\%$	5	20	4	6	$4.623 \pm 3\,\%$
8	10	8	4	$2.115 \pm 5\,\%$	6	20	4	6	$11.82 \pm 4\,\%$
1	10	8	6	$31.75 \pm 1\,\%$	7	20	4	6	$4.752 \pm 7\,\%$
2	10	8	6	$15.58 \pm 3\,\%$	8	20	4	6	$7.400 \pm 1\,\%$

Capítulo 6

Conclusiones y trabajo futuro

En este trabajo se abordó el problema de satisfacción para lógicas modales, así como sus principales aplicaciones en ámbitos como bases de datos [10], [11] y sistemas conscientes del contexto [28]. Posteriormente, se analizaron los algoritmos de satisfacción para lógicas modales existentes en el estado del arte.

Con base en lo anterior, se seleccionó la lógica multimodal K con inversas, debido a que las modalidades inversas incrementan la expresividad del lenguaje al tratar de representar relaciones en el sentido inverso. Por ejemplo, cuando se intenta razonar sobre el tiempo, es intuitivo que el pasado es la relación inversa del futuro y viceversa. Además, múltiples modalidades posibilitan razonar en problemas donde hayan distintos agentes o entidades interactuando en un sistema.

Para la construcción del algoritmo se especificó la propiedad de modelo de árbol y se realizó su demostración. El objetivo del algoritmo consiste en construir una estructura sintáctica de árbol, de ser posible, y responder al problema de satisfacción para esta lógica. Por lo que este teorema fungió de base para determinar si una fórmula se satisface o no bajo una estructura con forma de árbol.

El algoritmo se diseñó e implementó de manera eficiente utilizando algunas técnicas de optimización como la reutilización de memoria mediante el uso de estructuras dinámicas. Adicionalmente, se mantuvo una esencia iterativa, en lugar de una recursiva, y se comenzó a construir el árbol tomando en consideración la información que aporta la fórmula de entrada. Esto último con el objetivo de seleccionar los nodos candidatos de manera progresiva, además de emplear un enfoque de construcción de arriba a abajo.

Se realizaron experimentos con la implementación del algoritmo, con el objetivo de evaluar su rendimiento ante fórmulas tanto sintéticas como lo más cercanas a las que existen en la realidad, mediante conjuntos de fórmulas conocidos en la literatura, como lo es el TANCS2000. Dichos resultados son prometedores debido a que el algoritmo es capaz de obtener una respuesta en un tiempo razonable.

No obstante, aún hay trabajo futuro por realizar. Por ejemplo, es posible utilizar otro tipo de representación que permita realizar operaciones booleanas más eficientes, como los diagramas de decisión binarios [34]. Este tipo de diagramas mostraron un rendimiento superior en el trabajo mencionado, especialmente cuando se manejan fórmulas con modalidades profundas y que son de interés debido a que las modalidades diamante incrementan la complejidad de los modelos.

Los diagramas de decisión binarios serían de utilidad al momento de identificar contradicciones proposicionales en la función siguiente del algoritmo, cuando se intentan generar los nodos hijos de un nodo actual. Adicionalmente, ya existen implementaciones en el lenguaje de programación Go, lo que permite que estas estructuras se puedan emplear a nivel de implementación de manera inmediata.

Otra técnica de optimización que ha surgido recientemente es el uso de algoritmos de aprendizaje automatizado, debido a que existen trabajos donde han obtenido buenos resultados, por ejemplo, en probadores automáticos de teoremas [5], [21]. Estos trabajos emplean algoritmos de aprendizaje por refuerzo y aprendizaje profundo respectivamente.

Este algoritmo de satisfacción podría ser útil en la generación de conjuntos de datos de entrenamiento que sirva de entrada para este tipo de algoritmos de aprendizaje automatizado, ya que el algoritmo presentado en este trabajo es capaz de retornar una estructura sintáctica de árbol para fórmulas que se satisfacen.

Uno de los objetivos consistiría en determinar con cierta probabilidad el siguiente nodo a construir o probar, dado el conjunto *lean* de la fórmula, al momento de la construcción de los posibles arboles sintácticos. Esto podría mejorar la eficiencia del algoritmo especialmente cuando existen disyunciones, ya que ocasiona que haya múltiples nodos por probar, y usualmente hay caminos que son más cortos que otros o que no poseen contradicciones.

Para llevar a cabo la generación de un conjunto de datos, se tendría que construir de manera sintética un conjunto grande de formulas modales aleatorias de entrada, de manera que sean lo más representativas, evitando redundancias y sesgos, como ilustración, que exista variedad en cuanto al número de variables proposicionales, operadores modales, profundidad modal y operadores booleanos.

Adicionalmente, como los modelos de aprendizaje automático trabajan con números a su entrada, habría la necesidad de encontrar una forma de codificar la estructura sintáctica de árbol a una cadena de caracteres lineal manteniendo toda la información que el árbol proporciona.

Por otro lado, la intención de este trabajo es fungir de base en la creación de algoritmos para lógicas más expresivas, como lo es el cálculo μ. Donde también existen las modalidades inversas [26] e inclusive operadores especiales como los de punto fijo [16], de conteo [6],

entre otros.

El hecho de considerar nuevos operadores, implicaría tener que comprobar que la propiedad de modelo de árbol se mantiene en la nueva lógica. Después de realizar la prueba de la propiedad, se podría modificar el algoritmo para trabajar acorde con los nuevos operadores. Si, por ejemplo, se consideran operadores de punto fijo, el principal desafío se encuentra en el manejo de la recursividad, ya que habría que tratar de mantener un enfoque iterativo.

Finalmente, se planea aplicar el algoritmo en el ámbito de la verificación formal. Como ilustración, existe un trabajo en [29] que extiende la lógica temporal para la especificación y razonamiento de sistemas de cómputo, a través de una técnica de abstracción de predicados [15]. Esta técnica permite modelar sistemas de transición y representar un programa mediante predicados.

Esto es una muestra de como las lógicas modales pueden impactar en la verificación formal de sistemas de cómputo. Especialmente este proceso toma relevancia en sistemas críticos como plantas nucleares, sistemas de aviación, equipo médico, entre otros, donde los recursos o inclusive vidas pueden verse afectadas.

Referencias

[1] P. Abate y R. Goré, "The Tableau Workbench," *Electronic Notes in Theoretical Computer Science*, vol. 231, págs. 55-67, 2009.

[2] P. Abate, R. Goré y F. Widmann, "Cut-free single-pass tableaux for the logic of common knowledge," en *Workshop on Agents and Deduction at TABLEAUX*, vol. 2007, 2007.

[3] P. Abhinav, A. Bhat, C. T. Joseph y K Chandrasekaran, "Concurrency Analysis of Go and Java," en *2020 5th Int. Conf. Computing, Communication and Secur. (ICCCS)*, 2020, págs. 1-6.

[4] P. Balsiger y A. Heuerding, "Comparison of Theorem Provers for Modal Logics — Introduction and Summary," en *Automated Reasoning with Analytic Tableaux and Related Methods*, H. de Swart, ed., Springer Berlin Heidelberg, 1998, págs. 25-26.

[5] K. Bansal, S. Loos, M. Rabe, C. Szegedy y S. Wilcox, "HOList: An Environment for Machine Learning of Higher Order Logic Theorem Proving," en *Proc. 36th Int. Conf. Mach. Learn.*, K. Chaudhuri y R. Salakhutdinov, eds., vol. 97, PMLR, 2019, págs. 454-463.

[6] E. Bárcenas, "A counting logic for trees," *Computación y Sistemas*, vol. 19, n.º 2, págs. 407-422, 2015.

[7] T. Caridroit, J.-M. Lagniez, D. L. Berre, T. de Lima y V. Montmirail, "A SAT-based approach for solving the modal logic S5-satisfiability problem," en *Proc. AAAI Conf. Artif. Intell.*, vol. 31, 2017.

[8] P. Costanza, C. Herzeel y W. Verachtert, "A comparison of three programming languages for a full-fledged next-generation sequencing tool," *BMC Bioinformatics*, vol. 20, 2019.

[9] M. Davis, G. Logemann y D. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, n.º 7, págs. 394-397, 1962.

[10] A. A. Dixit y P. G. Kolaitis, "A SAT-Based System for Consistent Query Answering," en *Theory and Appl. of Satisfiability Testing – SAT 2019*, I. J. Mikoláš y Lynce, eds., Springer International Publishing, 2019, págs. 117-135.

[11] A. A. Dixit y P. G. Kolaitis, "CAvSAT: Answering Aggregation Queries over Inconsistent Databases via SAT Solving," en *Proc. 2021 Int. Conf. Management Data*, Association for Computing Machinery, 2021, págs. 2701-2705.

[12] E. Giunchiglia, A. Tacchella y F. Giunchiglia, "SAT-Based Decision Procedures for Classical Modal Logics," *Journal of Automated Reasoning*, vol. 28, n.º 2, págs. 143-171, 2002.

[13] F. Giunchiglia y R. Sebastiani, "Building Decision Procedures for Modal Logics from Propositional Decision Procedures: The Case Study of Modal K(m)," *Information and Computation*, vol. 162, n.º 1-2, págs. 158-178, 2000.

[14] V. Goranko y M. Otto, "Model theory of modal logic," en *Handbook of Modal Logic*, P. Blackburn, J. V. Benthem y F. Wolter, eds., vol. 3, Elsevier, 2007, cap. 5, págs. 249-329.

[15] S. Graf y H. Saidi, "Construction of abstract state graphs with PVS," en *Comput. Aided Verification*, Springer Berlin Heidelberg, 1997, págs. 72-83.

[16] K. Hoder, N. Bjørner y L. de Moura, "μZ–An Efficient Engine for Fixed Points with Constraints," en *Comput. Aided Verification*, G. Gopalakrishnan y S. Qadeer, eds., Springer Berlin Heidelberg, 2011, págs. 457-462.

[17] I. Horrocks, "The FaCT System," en *Automated Reasoning with Analytic Tableaux and Related Methods*, H. de Swart, ed., Springer Berlin Heidelberg, 1998, págs. 307-312.

[18] I. Horrocks y P. F. Patel-Schneider, "Comparing Subsumption Optimizations," en *Proc. Int. Workshop Description Logics*, E. Franconi, G. D. Giacomo, R. M. MacGregor, W. Nutt y C. A. Welty, eds., vol. 11, 1998.

[19] U. Hustadt y B. Konev, "TRP++ 2.0: A Temporal Resolution Prover," en *Automated Deduction – CADE-19*, F. Baader, ed., Springer Berlin Heidelberg, 2003, págs. 274-278.

[20] U. Hustadt y R. A. Schmidt, "MSPASS: Modal Reasoning by Translation and First-Order Resolution," R. Dyckhoff, ed., Springer Berlin Heidelberg, 2000, págs. 67-71.

[21] C. Kaliszyk, J. Urban, H. Michalewski y M. Olšák, "Reinforcement learning of theorem proving," en *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, Curran Associates Inc., 2018, págs. 8836-8847.

[22] S. A. Kripke, "Semantical considerations on modal logic," *Acta philosophica fennica*, vol. 16, págs. 83-94, 1963.

[23] R. E. Ladner, "The Computational Complexity of Provability in Systems of Modal Propositional Logic," *SIAM Journal on Computing*, vol. 6, n.º 3, págs. 467-480, 1977.

[24] J. Li, S. Zhu, G. Pu y M. Y. Vardi, "SAT-Based Explicit LTL Reasoning," en *Hardware and Softw.: Verification and Testing*, N. Piterman, ed., Springer International Publishing, 2015, págs. 209-224.

[25] Y. Limón, E. Benítez-Guerrero, E. Bárcenas, G. Molero-Castillo y A. Velázquez-Mena, "A Satisfiability Algorithm For The Mu-Calculus For Trees With Presburger Constraints," en *7th Int. Conf. Softw. Eng. Res. and Innov. (CONISOFT 2019)*, 2019, págs. 72-79.

[26] Y. Limón, E. Bárcenas, E. Benítez-Guerrero, G. M. Castillo y A. Velázquez-Mena, "Mu-Calculus Satisfiability with Arithmetic Constraints," *Programming and Computer Software*, vol. 46, págs. 503-510, 2020.

[27] Y. Limón, E. Bárcenas, E. Benítez-Guerrero y C. Mezura-Godoy, "Towards a Reasoning Model for Context-aware Systems: Modal Logic and the Tree Model Property.," *Research in Computing Science*, vol. 99, págs. 9-18, 2015.

[28] Y. Limón, E. Bárcenas, E. Benítez-Guerrero et al., "Consistency checking of attention aware systems.," en *12th Latin Amer. Workshop on New Methods of Reasoning*, 2019, págs. 13-23.

[29] A. Lisitsa e I. Potapov, "Temporal logic with predicate /spl lambda/-abstraction," en *12th Int. Symposium on Temporal Representation and Reasoning (TIME'05)*, 2005, págs. 147-155.

[30] E. Lorini y F. Romero, "SAT for Epistemic Logic Using Belief Bases," en *Eng. Multi-Agent Syst.*, L. A. Dennis, R. H. Bordini e Y. Lespérance, eds., Springer International Publishing, 2020, págs. 235-245.

[31] F. Massacci y F. M. Donini, "Design and Results of TANCS-2000 Non-classical (Modal) Systems Comparison," en *Automated Reasoning with Analytic Tableaux and Related Methods*, R. Dyckhoff, ed., Springer Berlin Heidelberg, 2000, págs. 52-56.

[32] D. Medina-Martínez, E. Bárcenas, G. Molero-Castillo, A. Velázquez-Mena y R. Aldeco-Pérez, "A satisfiability algorithm for multi-modal logic with converse," en *11th Int. Conf. Softw. Eng. Res. and Innov. (CONISOFT 2023)*, 2024.

[33] V. Montmirail, "Practical resolution of satisfiability testing for modal logics.(Résolution pratique du test de cohérence en logiques modales).," Tesis doct., Artois Univ., Lens, France, 2018.

[34] G. Pan, S. Ulrike y M. Y. Vardi, "BDD-based decision procedures for the modal logic K," *Journal of Applied Non-Classical Logics*, vol. 16, n.º 1-2, págs. 169-207, 2006.

[35] G. Pan y M. Y. Vardi, "Optimizing a BDD-Based Modal Solver," en *Int. Conf. on Automated Deduction*, F. Baader, ed., Springer Berlin Heidelberg, 2003, págs. 75-89.

[36] P. Patel-Schneider, "System Description: DLP," en *Int. Conf. on Automated Deduction*, D. McAllester, ed., Springer Berlin Heidelberg, 2000, págs. 297-301.

[37] N. Ryo, *Vartan module - Go Packages*, fecha de acceso: 3 de Octubre, 2022. dirección: `https://pkg.go.dev/github.com/nihei9/vartan`.

[38] R Sebastiani y M Vescovi, "Automated reasoning in modal and description logics via SAT encoding: the case study of K (m)/ALC-satisfiability," *Journal of Artificial Intelligence Research*, vol. 35, págs. 343-389, 2009.

[39] R. Sebastiani y M. Vescovi, "Encoding the Satisfiability of Modal and Description Logics into SAT: The Case Study of K(m)/$\mathcal{ALC}$," en *Theory and Appl. of Satisfiability Testing - SAT 2006*, A. Biere y C. P. Gomes, eds., Springer Berlin Heidelberg, 2006, págs. 130-135.

[40] S. Tobies, "PSPACE Reasoning for Graded Modal Logics," *Journal of Logic and Computation*, vol. 11, n.º 1, págs. 85-106, 2001.

[41] S. Uwe y J. Torán, *The Satisfiability Problem: Algorithms and Analyses*. Lehmanns media, 2013, vol. 3.

[42] M. Y. Vardi, "Why is modal logic so robustly decidable?" *Descriptive complexity and finite models*, vol. 31, págs. 149-184, 1996.

[43] H. Zhang, "SATO: An efficient propositional prover," en *Int. Conf. on Automated Deduction*, W. McCune, ed., Springer Berlin Heidelberg, 1997, págs. 272-275.