SIMETRÍAS EN RAZONAMIENTO AUTOMÁTICO

ALEJANDRO EZEQUIEL ORBE

EL CASO DE LAS LÓGICAS MODALES Y SATISFACIBILIDAD MODULO TEORÍAS

Tesis de Doctorado
Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba

Director: Carlos Areces

Marzo 2014

A Ceci y Nacho.

A mi vieja y mi viejo.

A mis hermanos Rubén, Fernando y Roberto.

RESUMEN

Muchos problemas de la vida real presentan simetrías. Informalmente podemos definir una simetría de un objeto discreto como una permutación de sus componentes que mantiene invariante el objeto o algún aspecto del mismo.

En razonamiento matemático es una práctica común utilizar las simetrías de un problema para reducir la complejidad del razonamiento ya que uno puede analizar en detalle uno de los casos simétricos y generalizar el resultado a los otros.

En el contexto de razonamiento automático, se puede definir como simetría de una fórmula a una permutación de sus variables (o literales) que mantenga su estructura y sus modelos. Si somos capaces de reconocer que una fórmula posee simetrías podemos guiar al algoritmo de búsqueda para que sólo busque soluciones en las partes no simétricas del espacio de búsqueda. Durante los últimos años, se ha investigado exhaustivamente sobre simetrías (como detectarlas y como utilizarlas), principalmente en los contextos de SAT (lógica proposicional) [Sakallah, 2009] y de CSP (Constraint Satisfaction Problem) [Gent *et al.*, 2006]. También, aunque en menor medida, se ha investigado el uso de simetrías en otras lógicas [Audemard *et al.*, 2004; Audemard *et al.*, 2007b; Audemard, 2002; Audemard *et al.*, 2006].

En esta tesis investigamos el uso de simetrías en el contexto de lógicas modales y de satisfacibilidad modulo teorías (SMT).

En primer lugar desarrollamos un marco teórico para utilizar las simetrías de una formula modal. Este marco teórico nos permite trasladar resultados existentes en otras lógicas a las lógicas modales. Debido a la gran variedad de lógicas modales existentes, desarrollamos el marco teórico utilizando el marco de modelos modales coinductivos [Areces and Gorín, 2010]. Esto nos permite obtener resultados generales que pueden ser aplicados a toda lógica modal que pueda ser representada en dicho marco.

A continuación definimos algoritmos de detección de simetrías en fórmulas en forma normal conjuntiva. Estos algoritmos están basados en el problema de detección de automorfismos de un grafo coloreado [Fortin, 1996], el cual se crea a partir de la fórmula y permiten la detección de simetrías de literales de una formula. Luego una variante de este algoritmo es presentado para SMT el cual permite detectar simetrías de símbolos no interpretados en una formula. Todos los algoritmos son evaluados experimentalmente, demostrando la existencia de un gran número de simetrías en las fórmulas investigadas.

Finalmente, presentamos una técnica para utilizar información de simetrías en algoritmos de satisfacibilidad modal. La misma consiste en la incorporación de una condición de bloqueo adicional a un cálculo de tableaux modal de forma de que solo se aplique la regla $(\Diamond)$ a aquellas fórmulas que no contengan una fórmula simétrica a la cual ya se haya aplicado la regla. Probamos la completitud del cálculo obtenido y evaluamos experimentalmente esta técnica en distintos conjuntos de fórmulas modales.

PUBLICACIONES

Esta tesis está basada en las siguientes publicaciones:

- C. Areces, and E. Orbe. *Dealing with Symmetries in Modal Tableaux*. In Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX). Nancy, Francia. 2013.
- C. Areces, D. Deharbe, P. Fontaine, and E. Orbe. *SyMT: finding symmetries in SMT formulas*. 11th International Workshop on Satisfiability Modulo Theories (SMT 2013), Helsinki, Finlandia. 2013
- C. Areces, G. Hoffmann, E. Orbe. *Symmetries in Modal Logics*. Post-Proceedings of the Seventh Workshop on Logical and Semantic Frameworks, with Applications, Rio de Janeiro, Brasil, vol. 113. 2013.
- C. Areces, G. Hoffmann, E. Orbe. *Symmetries in Modal Logics: A Coinductive Approach*. Seventh Workshop on Logical and Semantic Frameworks, with Applications, Rio de Janeiro, Brasil. 2012.

RECONOCIMIENTOS

En primer lugar quiero agradecer a mi director, Carlos Areces, por guiarme durante estos últimos tres años, y por los miles de consejos que me ha dado tan generosamente. Estoy seguro que dirigirme no ha sido una tarea fácil, y que ha requerido mucha paciencia por parte de Carlos. Muchas gracias Carlos.

Siempre estaré agradecido con Gabriel Infante-Lopez. Gabriel fue quien me dió la posibilidad de entrar al mundo académico y quien me motivó a perseguir el doctorado. Gracias por las charlas y por tu amistad Gabriel.

La academia también me dió la chance de conocer a algunos de mis más queridos amigos. En particular quiero agradecer a Martín, Camper y Diego, por los muchos asados que compartimos y las charlas sobre la ciencia y la vida. Tampoco me quiero olvidar de Franco, Chun, Guillaume y Raúl, gracias chicos por ayudarme durante estos cinco años.

Esta tesis no hubiese sido posible sin el apoyo de mi familia. Gracias Vieja y Viejo por todo, por estar siempre presentes cuando los necesité y por todo el cariño que me han dado. Viejo, se que, donde quieras que estes, debes estar muy orgulloso. Te extraño mucho. Mis hermanos Rubén, Fernando y Roberto, también han sido vitales para lograr esto. Uds, como siempre, han sido una fuente constante de consejos y la verdad que no se que haría sin uds.

Finalmente, quiero agradecer a los amores de mi vida: Ceci y Nacho. Ustedes son la fuente de todas mis sonrisas y los que han tenido que lidiar conmigo cuando las cosas nos salían como uno las esperaba. Todo esto no hubiese sido posible sin ustedes. Los amo.

Córdoba, 13/02/2014.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Parte I

INTRODUCCIÓN

"Symmetry as wide or as narrow as you may define its meaning, is one idea by which man through the ages has tried to comprehend and create order, beauty and perfection."

Hermann Weyl. *Symmetry*. 1952

1

SIMETRÍAS EN RAZONAMIENTO AUTOMÁTICO

La noción de simetría es una noción familiar: reconocemos una simetría cuando la vemos. El término *simetría* es utilizado en un sentido muy amplio, no solo se utiliza como una noción matemática, sino también como un concepto que sirve de puente entre diversas disciplinas, que, desde que fue utilizado por primera vez por los antiguos griegos, se ha ido transformando en un proceso largo y lento (ver [Brading and Castellani, 2013; Darvas, 2007; Hon and Goldstein, 2008] para más detalles sobre la evolución del concepto de simetría).

En general, podemos hablar de la simetría de un objeto cuando "bajo algún tipo de transformación al menos una propiedad del objeto permanece invariante". Sin embargo, es la definición de simetría basada en teoría de grupos la que se ha mostrado más útil, y sobre la cual trabajaremos en esta tesis.

En el contexto de la teoría de grupos, podemos definir informalmente una simetría como "invariancia bajo un grupo específico de transformaciones". Esta definición revela una estrecha conexión entre los conceptos de simetría, equivalencia y grupo: El grupo de simetrías induce una partición de los elementos que se intercambian en clases de equivalencia, de manera tal que, los elementos intercambiados por una simetría están relacionados mediante una relación de equivalencia, formando de este modo una clase de equivalencia.

El concepto de simetría tiene muchos usos. No sólo podemos estudiar las propiedades simétricas de un objeto (geométrico, matemático, etc.) para comprender su comportamiento, sino que también podemos derivar consecuencias específicas en relación con el objeto bajo estudio en función de sus propiedades de simetría, utilizando "argumentos basados en simetría".

Este tipo de argumentos es de uso común en el razonamiento matemático. Las pruebas matemáticas suelen afirmar que cierta suposición puede ser hecha "sin pérdida de generalidad". Esta frase sugiere que, aunque a primera vista sólo se demuestra el teorema para un caso restringido, el argumento justifica el teorema en general [Harrison, 2009].

La idea que subyace en este tipo de argumentos es que los problemas estructuralmente similares (problemas simétricos) deben tener soluciones similares, es decir, que una solución para dicho problema debe respetar las simetrías del mismo.

Por lo tanto, si podemos identificar las simetrías de un problema, podaríamos utilizarlas para reducir la dificultad del razonamiento analizando en detalle sólo uno de los problemas simétricos (casos) y luego generalizando el resultado a los demás. Esto es exactamente lo que tratamos de hacer cuando utilizamos las simetrías de un problema en el contexto del razonamiento automático.

Muchas clases de problemas, en particular, aquellas derivadas de aplicaciones concretas, presentan un gran número de simetrías. En razonamiento automático, la presencia de simetrías en el espacio de búsqueda de un problema puede aumentar la dificultad para encontrar una solución al forzar al algoritmo de búsqueda a explorar subespacios simétricos que no contienen soluciones. Si somos capaces

de reconocer que existen estas simetrías, podemos utilizarlas para direccionar al algoritmo de búsqueda para que busque soluciones sólo en regiones no simétricas del espacio de búsqueda, reduciendo así la dificultad [Sakallah, 2009].

Ahora, ¿qué queremos decir cuando hablamos de las "simetrías de un problema" en el contexto del razonamiento automático? Definiremos como *simetría* de un problema a una permutación de sus variables (o literales) que preserve su estructura (su forma sintáctica) y, por lo tanto, su conjunto de soluciones (su modelos).

Dependiendo de qué aspecto del problema se mantiene invariante, podemos clasificar una simetría como *semántica* o *sintáctica* [Benhamou and Sais, 1992]. Una simetría semántica es una propiedad intrínseca de la función y por lo tanto es independiente de cualquier representación particular, en otras palabras, es una permutación de las variables del problema que no cambia el valor de la función en virtud de cualquier asignación de valor a las variables. Una simetría sintáctica, por otro parte, corresponde a la representación algebraica específica de la función, es decir, es una permutación de las variables (o literales) que no cambia la representación sintáctica del problema. Una simetría sintáctica es una simetría semántica, pero lo inverso no siempre se cumple.

Mientras que el interés en las simetrías semánticas fue motivado principalmente por el deseo de optimizar el diseño de circuitos lógicos y para acelerar su verificación, el interés en las simetrías sintácticas surgió en el contexto de resolución de restricciones (constraint solving), en satisfacibilidad proposicional específicamente (SAT).

La primer referencia a los potenciales beneficios de utilizar simetrías sintácticas en el razonamiento lógico se atribuye a [Krishnamurthy, 1985] donde se introduce el *principio de simetría* (o *regla de simetría*) para fortalecer un sistema de pruebas basado en resolución para la lógica proposicional, y donde se demuestra que, con esta regla, es posible reducir el tamaño de las pruebas para ciertas clases de problemas matemáticos complejos (ej., rompecabezas, teorema de Ramsey y principio de los nidos de palomas – pigeonhole principle [Urquhart, 1999]).

La intuición que subyace al principio de simetría es que en el curso de una prueba matemática, a menudo se utiliza un elemento arbitrario de un conjunto como representante del conjunto, siempre que el conjunto sea simétrico, de modo que los argumentos subsiguientes se aplican del mismo modo a los demás elementos del conjunto. De forma similar, este principio reconoce que una tautología permanece invariante bajo ciertas permutaciones de nombres de variables y utiliza esa información para evitar repetir derivaciones de fórmulas intermedias que son meramente variantes permutacionales la una de la otra.

En realidad [Krishnamurthy, 1985] presenta dos reglas de inferencia: *la regla de simetría global* y *la regla de simetría local*. Dado un conjunto de cláusulas proposicionales Γ (es decir, un conjunto de conjuntos de literales proposicionales) y una permutación σ que mapea literales a literales, si derivamos una cláusula C a partir de Γ, la regla de simetría global nos permite inferir la cláusula $\sigma(C)$ siempre que σ sea una simetría de Γ (es decir, $\sigma(\Gamma) = \Gamma$). Por otro lado, la regla de simetría local nos permite inferir $\sigma(C)$ bajo la condición de que para cada cláusula A en Γ utilizada en la derivación de C, $\sigma(A)$ también esté en Γ.

La diferencia entre las dos reglas reside en el hecho de que la primer regla requiere que la permutación mantenga invariante todo el conjunto de cláusulas Γ,

mientras que la segunda sólo requiere que las cláusulas simétricas de las cláusulas utilizadas en una derivación pertenezcan al conjunto de cláusulas Γ. Por lo tanto, la regla de simetría local puede ser aplicable incluso en casos en que la regla de simetría global no lo es y, por lo tanto, es estrictamente más fuerte que la regla de simetría global [Arai and Urquhart, 2000].

A pesar de mostrar la utilidad de las reglas de simetría para algunos problemas [Krishnamurthy, 1985] no presenta algoritmo alguno para realizar la detección de simetrías de un conjunto de cláusulas, y tampoco se discute en detalle cómo los procedimientos de resolución deben utilizar las reglas de simetría.

La primer referencia sobre el uso de simetrías en un algoritmo de búsqueda se debe a [Brown *et al.*, 1989]. En este trabajo, el algoritmo de búsqueda utiliza las simetrías dinámicamente para restringir su búsqueda a la clases de equivalencia inducidas por las simetrías de la fórmula que se asume que son dadas al inicio de la ejecución.

En [Benhamou and Sais, 1992; Benhamou and Sais, 1994] se presentan un algoritmo de detección de simetrías y algoritmos de búsqueda que aprovechan la información de simetrías del problema. La detección de simetrías se realiza directamente en la fórmula de entrada derivando permutaciones de variables, que preserven la fórmula, de forma incremental. Una vez que las simetrías son detectadas, se muestra cómo utilizarlas en dos algoritmos de búsqueda: el algoritmo SLRI [Cubadda, 1988] (una variante del algoritmo de resolución SL [Kowalski and Kuehner, 1972]) y el algoritmo de evaluación semántica [Oxusoff, 1989].

En ambos algoritmos, la clave, para sacar provecho de las simetrías durante el proceso de deducción, es una propiedad que establece que un literal l tiene un modelo en un conjunto de cláusulas S (es decir, hay una valuación v que satisface S tal que $V(l) = true$) si y sólo si para cada literal simétrico l' (es decir, cada literal tal que $\sigma(l) = l'$, para una simetría σ) existe un modelo v' de S tal que $v(l') = true$. De esta forma, en el caso de SLRI, en lugar de refutar cada literal por separado, utilizando simetrías es posible refutar los literales simétricos de forma simultánea, por ejemplo, sea $C = \{l_1, l_2, \ldots, l_{i-1}, l_i, \ldots, l_n\}$ una cláusula, y supongamos que los literales $l_1, l_2, \ldots, l_{i-1}$ ya han sido refutados y que l_i es el literal que intentamos refutar. Si l_i es simétrico a uno de los literales ya refutados, entonces será refutado directamente utilizando esta propiedad de simetría. Cuanto más literales simétricos en una cláusula, mayor es el recorte que podemos hacer en el árbol de resolución. Del mismo modo, en evaluación semántica, si el algoritmo asigna el valor verdadero o falso a un literal y genera la cláusula vacía, entonces insertamos, en el modelo que se está construyendo, el valor opuesto al literal, junto con todos los demás opuestos de sus literales simétricos.

En [Crawford, 1992] se presenta un enfoque diferente para la detección de simetrías. En este trabajo se demuestra que el problema de detección de simetrías en fórmulas proposicionales se puede reducir polinomialmente al problema de detección de automorfismos en grafos coloreados y se presenta un algoritmo para crear un grafo coloreado a partir de una fórmula proposicional en forma normal conjuntiva (CNF). Luego, el grupo de simetrías (en realidad, un subgrupo del grupo de simetrías) de la fórmula es detectado mediante la detección del grupo de automorfismos del grafo resultante. [Crawford, 1992] también muestra como evaluar los efectos de la simetrías en el conjunto de asignaciones mediante el uso del Teorema

de Polya [Pólya and Read, 1987] para contar el número de clases de equivalencia inducidas en el conjunto de asignaciones por un conjunto de simetrías de una fórmula dada. Sin embargo ningún algoritmo práctico para utilizar las simetrías es presentado.

En [Crawford *et al.*, 1996] un método para utilizar las simetrías es presentado. En lugar de modificar el algoritmo de búsqueda para usar las simetrías, este método modifica el problema a resolver mediante la adición de un predicado de ruptura de simetría– symmetry breaking preadicate (SBP) – para cada simetría. Estos predicados están construidos de forma tal que son verdaderos en exactamente una de las asignaciones dentro de cada clase de equivalencia inducida por el grupo de simetrías; la fórmula resultante es equisatisfacible a la original. Los predicados se construyen de forma tal que son verdaderos sólo en el modelo más pequeño (el lex-líder), dentro de cada clase de equivalencia, con respecto a un orden lexicográfico definido sobre el conjunto de asignaciones a partir de un ordenamiento preestablecido en el conjunto de variables. Intuitivamente, podemos pensar en las asignaciones como números binarios, y en los predicados como realizando una comparación bit a bit entre dos asignaciones y seleccionado la menor de las dos.

Dado que, en general, romper todos las simetrías en un grupo de simetrías puede resultar en una fórmula exponencialmente más grande que la original (debido al tamaño del grupo de simetrías y a la presencia de cláusulas redundantes en los SBPs), los autores proponen romper sólo algunas de las simetrías mediante la poda de un *árbol de simetrías* construido a partir del grupo de simetrías. Sin embargo, esto no siempre evita cláusulas redundantes y tampoco asegura que la fórmula resultante sea sólo polinomialmente más grande que la original.

En [Aloul *et al.*, 2003b] se presentan mejoras al trabajo de [Crawford, 1992; Crawford *et al.*, 1996]. En particular, se introducen nuevas construcciones de grafos con las cuales es posible detectar más simetrías que con las construcciones de [Crawford, 1992; Crawford *et al.*, 1996]. También se presentan mejoras en la generación de los predicados de ruptura de simetrías. Al igual que en [Crawford *et al.*, 1996], en [Aloul *et al.*, 2003b] se explora la ruptura parcial de simetrías mediante la construcción de predicados que no sólo seleccionen las asignaciones más pequeñas respecto de cierto orden lexicográfico. Ya que los predicados se construyen para cada simetría, lo que proponen es realizar dicha construcción sólo para un conjunto de generadores del grupo de simetría de la fórmula [Seress, 1997]. Al romper solamente los generadores del grupo, no se rompen todas las simetrías, pero aún se logra una poda significativa en el espacio de búsqueda. Otra variante propuesta, es la forma en la cual se construyen los predicados, ya que propone formular los predicados en términos de los ciclos de un permutación, es decir, para cada ciclo individual en la simetría, en lugar de para todo el conjunto de variables. De esta forma se generan predicados más pequeños. Luego, en [Aloul *et al.*, 2003a; Aloul *et al.*, 2006], se presenta una construcción más sistemática y eficiente de los predicados. Esta construcción tiene en cuenta la estructura del ciclo de generadores, que normalmente involucra muy pocas variables, lo cual permite reducir drásticamente el tamaño de los predicados generados.

Un enfoque diferente para la utilización de las simetrías de una fórmula, se presenta en [Sabharwal, 2005]. En este trabajo se introduce una modificación al SAT solver zChaff [Moskewicz *et al.*, 2001] que le permite tomar decisiones sobre un

conjunto de variables simétricas en lugar de hacerlo para cada variable individualmente. Por ejemplo, dado un conjunto $\{x_1, x_2, \ldots, x_k\}$ de k variables simétricas, una $k+1$-decisión fija $x_1, \ldots, x_i$ en 0 y $x_{i+1}, \ldots, x_k$ en 1 para cada $i \in [0, k]$. Esto reduce el número de asignaciones parciales que potencialmente deben ser exploradas de 2^k a $k+1$. La identificación de las variables simétricas se supone que esta disponible a partir de descripciones de alto nivel del problema y son proporcionadas al solver como información adicional.

En [Benhamou *et al.*, 2010] se presenta una técnica que combina el razonamiento simétrico con aprendizaje de cláusulas en SAT solvers CDCL [Een and Sörensson, 2003]. La idea es mejorar el proceso de aprendizaje de cláusulas mediante la utilización de las simetrías del problema, de forma tal de poder aprender las cláusulas simétricas a la cláusula aprendida. Aquí el foco no está en guiar al algoritmo de búsqueda directamente, sino, en enriquecer un proceso complementario y así proporcionar más información al algoritmo.

Hasta el momento, hemos proporcionado una breve reseña de los trabajos más importantes en el área de simetrías aplicadas a la satisfacibilidad proposicional. Sin embargo, existe un gran cúmulo de investigación sobre este tema, al cual, en líneas generales, podemos agrupar en dos categorías diferentes: ruptura dinámica de simetrías (dynamic symmetry breaking) [Brown *et al.*, 1989; Benhamou and Sais, 1992; Benhamou and Sais, 1994; Sabharwal, 2005] y ruptura estática de simetrías (static symmetry breaking) [Crawford, 1992; Crawford *et al.*, 1996; Aloul *et al.*, 2003b; Aloul *et al.*, 2003a; Aloul *et al.*, 2006].

En ruptura dinámica de simetrías la idea es utilizar las simetrías para restringir dinámicamente el espacio de búsqueda, mientras que en la ruptura estática de simetrías, las mismas se eliminan del problema antes de utilizar el solver, es decir, en una etapa de preprocesamiento. El primer enfoque depende del solver utilizado, pero puede aprovechar las simetrías que emerjan durante la búsqueda, mientras que el segundo enfoque se puede utilizar con cualquier solver. A pesar de sus diferencias ambos comparten el mismo objetivo: identificar las ramas simétricas en el espacio de búsqueda y guiar al solver lejos de aquellas ramas simétricas ya exploradas.

El uso de simetrías también ha sido ampliamente investigado, y con éxito, en otros dominios. En Constraint Satisfaction Problem (CSP) [Gent *et al.*, 2006] se realizado mucho trabajo sobre simetrías, en particular, enfocado en proporcionar definiciones correctas de lo que exactamente es una simetría de un problema CSP [Cohen *et al.*, 2005]. En Integer Programming, algoritmos han sido adaptados para explotar grandes grupos de simetría [Margot, 2002; Margot, 2003]. En Planning, intentos de integrar razonamiento simétrico a un planificador se han hecho en [Fox and Long, 1999; Fox and Long, 2002], y ha sido señalado que la detección de simetrías durante la búsqueda, y la presencia de çasi-simetrías", es un tópico muy importante en el área de planning debido a la naturaleza de los problemas de planificación.

También se han realizado esfuerzos sustanciales en Model Checking [Clarke *et al.*, 1996; Ip and Dill, 1996; Sistla *et al.*, 2000; Bošnački *et al.*, 2002]. Estos trabajos asumen que los usuarios reconocen las simetrías, y se han limitado a lidiar con casos sencillos de simetrías.

En el área de Quantified Boolean Formulas (QBF), en [Audemard *et al.*, 2004], se presenta un enfoque híbrido que integra fórmulas QBF y predicados de ruptura

de simetría proposicionales. En el contexto de QBF, los predicados proposicionales puros no pueden ser añadidos conjuntivamente a la fórmula original, porque, ya que se cuantifican las variables, se pueden obtener cláusulas con todas sus variables cuantificadas universalmente, y en consecuencia la QBF no puede ser válida. Para evitar tal inconveniente, los autores proponen considerar los predicados como independiente de la fórmula QBF. Un solver híbrido es diseñado entonces para trabajar simultáneamente con la fórmula QBF y los predicados de ruptura de simetrías. También se presenta un algoritmo de detección de simetrías para fórmulas QBF que extiende los algoritmos de detección basados en grafos utilizados en problemas proposicionales. Una técnica más práctica de generación de predicados de ruptura de simetrías para QBF se presenta en [Audemard *et al.*, 2007a; Audemard *et al.*, 2007b].

1.1 SIMETRÍAS EN LAS LÓGICAS MODALES Y EN SATISFACIBILIDAD MODULO TEORÍAS

En el caso de las lógicas modales, se ha investigado la forma de utilizar simetrías en el área de chequeo de modelos para la lógica temporal LTL [Clarke *et al.*, 1996; Donaldson and Miller, 2005; Miller *et al.*, 2006; Donaldson, 2007], la lógica temporal-epistémica [Cohen *et al.*, 2009]. Sin embargo, hasta donde conocemos, el uso de simetrías en satisfacibilidad y demostración automática de teoremas para lógicas modales permanece en gran parte inexplorado.

La situación en SMT es, de alguna manera, mejor, ya que se han realizado intentos para utilizar las simetrías de un problema en chequeo de modelos basado en SMT [Audemard *et al.*, 2002] y en SMT solving [Roe, 2006; Déharbe *et al.*, 2011] (véase el Capítulo 4.1 para más detalles). En particular, en [Déharbe *et al.*, 2011] un algoritmo para la detección y la ruptura de simetrías es presentado y puesto en práctica en el solver veriT [Bouton *et al.*, 2009]. Sin embargo, la técnica de detección de simetría es bastante heurística y la técnica de ruptura de simetrías sólo funciona para ciertas clases de problemas.

En esta tesis, tenemos dos objetivos principales. En primer lugar, nuestro objetivo es proporcionar un presentación completa sobre cómo utilizar simetrías en las lógicas modales, proporcionando la base teórica para utilizarlas, las algoritmos para detectarlas, y técnicas para utilizarlas en un solver modal.

En segundo lugar, para SMT, nos centramos en mejorar el trabajo en [Déharbe *et al.*, 2011], mediante el desarrollo de una técnica de detección general que sea capaz de detectar más simetrías que las detectadas por las técnicas conocidas hasta ahora.

2

SIMETRÍAS EN LAS LÓGICAS MODALES

En el Capítulo 1 vimos que las simetrías han sido extensamente investigadas en satisfacibilidad para lógica proposicional y en otras lógicas. Sin embargo, todavía permanecen largamente inexploradas en el contexto de razonamiento automático para las lógicas modales. En este capítulo intentamos "completar" este vacío mediante el desarrollo de los fundamentos teóricos que permitan la utilización de las simetrías en las lógicas modales. Lo haremos gradualmente, presentando primero las definiciones y resultados para la lógica modal básica y luego extendiendo los mismos al marco de los modelos modales coinductivos.

Comenzamos presentando el lenguaje modal básico y mostrando como las permutaciones de literales pueden definir simetrías en fórmulas modales en forma clausal, y como, estas simetrías, comparten propiedades similares a las simetrías definidas en el caso proposicional (Sección 2.1). Luego extendemos los resultados a un conjunto más amplio de lenguajes modales (Sección 2.2). Primero presentamos el marco de los modelos modales coinductivos (Sección 2.2.1). Luego, generalizamos la noción de simetría a fórmulas modales en forma clausal para diferentes lógicas modales incluyendo la lógica modal básica sobre diferentes clases de modelos (e.j., modelos reflexivos, lineales o transitivos), y lógicas con operadores adicionales (e.j., operador universal y operadores híbridos), usando el marco que nos proveen los modelos modales coinductivos (Sección 2.2.2). Finalmente, para los casos en los cuales la lógica posee la propiedad de modelo arbolar (tree model property), desarrollamos una noción más flexible de simetría que nos permite encontrar simetrías que no podrían ser encontradas de otra forma (Sección 2.2.3).

Los resultados presentados en este capítulo fueron publicados en [Areces *et al.*, 2012].

2.1 SIMETRÍAS EN LA LÓGICA MODAL BÁSICA

En lo que sigue, asumimos conocimientos básicos sobre las lógicas modales clásicas (por mas detalles ver [Blackburn *et al.*, 2001; Blackburn *et al.*, 2006]).

Definición 2.1 (Sintáxis)**.** *Sea* $\mathsf{PROP} = \{p_1, p_2, \ldots\}$ *un conjunto contable infinito de variables proposicionales y* $\mathsf{MOD} = \{m, m'', \ldots\}$ *un conjunto de símbolos de modalidades. Denominamos como* signatura modal *al par* $\mathcal{S} = \langle \mathsf{PROP}, \mathsf{MOD} \rangle$*. El conjunto de las fórmulas modales básicas* FORM *sobre la signatura* $\mathcal{S}$ *se define como*

$$\mathsf{FORM} ::= p \mid \neg\varphi \mid \varphi \vee \psi \mid [m]\varphi,$$

donde $p \in \mathsf{PROP}$, $m \in \mathsf{MOD}$, y $\varphi, \psi \in \mathsf{FORM}$. $\top$ y $\bot$ *representan una tautología arbitraria y la contradicción, respectivamente. También utilizaremos los conectivos clásicos como* $\wedge, \rightarrow$ *definidos de la forma usual. Por cada* $m \in \mathsf{MOD}$ *tenemos un* operador dual $\langle m \rangle$ *(diamante) definido como* $\langle m \rangle \varphi = \neg[m]\neg\varphi$*. Cuando* MOD *es un singleton, es decir, en el caso mono-modal, simplemente escribimos* $\Box$ y $\Diamond$*.*

Dado que estamos interesados en las simetrías sintácticas de las fórmulas modales, normalizaremos su representación sintáctica.

Definición 2.2 (Literales y CNF modal)**.** *Un* literal proposicional *l es una variable proposicional p o su negación $\neg p$. El conjunto de literales sobre* PROP *es* $\mathsf{PLIT} = \mathsf{PROP} \cup \{\neg p \mid p \in \mathsf{PROP}\}$. *Una fórmula modal está en* forma normal conjuntiva modal (CNF modal) *si es una conjunción de cláusulas en CNF modal. Una* cláusula en CNF modal *es una disyunción de literales proposicionales y modales. Un* literal modal *es una fórmula de la forma $[m]C$ o $\neg[m]C$ donde C es una cláusula en CNF modal.*

Decimos que dos fórmulas son *equisatisfacibles* si una fórmula es satisfacible si y solo si la otra también es satisfacible. Toda fórmula modal puede ser transformada en una fórmula en CNF modal equisatisfacible en tiempo polinomial [Patel-Schneider and Sebastiani, 2003].

A partir de ahora, asumiremos que todas las fórmulas están en CNF modal. También representaremos a las fórmulas en CNF modal como un conjunto de cláusulas en CNF modal interpretado conjuntivamente, y a una cláusula en CNF modal como un conjunto de literales proposicionales y modales, interpretado disyuntivamente. De esta forma, al utilizar la representación como conjuntos, podemos ignorar el orden y la multiplicidad de las cláusulas y literales que aparecen en la fórmula. Esto será importante al momento de definir las simetrías.

Definición 2.3 (Profundidad Modal)**.** *La* profundidad modal *de una fórmula φ (notación $md(\varphi)$) es una función que va de fórmulas a los números naturales definida de la siguiente forma:*

$$\begin{aligned} md(p) &= 0, \text{ para } p \in \mathsf{PROP} \\ md(\neg\varphi) &= md(\varphi) \\ md(\varphi \vee \psi) &= \text{máx}\{md(\varphi), md(\psi)\} \\ md([m]\varphi) &= 1 + md(\varphi). \end{aligned}$$

En otras palabras, la profundidad modal de una fórmula es el máximo anidamiento de operadores modales, e.j., $md(\langle m\rangle(p \wedge q \wedge p) \wedge [m][m]\neg r) = 2$, y es una medida de la complejidad de la fórmula.

En este trabajo utilizaremos la semántica relacional de la lógica modal básica, también conocida como semántica de Kripke [Kripke, 1959; Kripke, 1963].

Definición 2.4 (Modelos)**.** *Un modelo (o modelo de Kripke) $\mathcal{M}$ es una tripla $\mathcal{M} = \langle W, \{R^m\}_{m\in\mathsf{MOD}}, V\rangle$ donde:*

i) W, el dominio, *es un conjunto no vacío. Los elementos de W se denominan* puntos, estados, mundos, *etc.*

ii) Cada R^m es una relación binaria sobre W.

iii) V, la valuación, *es una función que asigna a cada elemento $w \in W$ un subconjunto $V(w) \subseteq \mathsf{PROP}$. Informalmente, podemos pensar a $V(w)$ como el conjunto de variables proposicionales que son verdaderas en w.*

Nota. De ahora en adelante trabajaremos con el caso mono-modal. Todos los resultados presentados se extienden naturalmente al caso multi-modal.

Definición 2.5 (Modelos Puntuados)**.** *Un* modelo puntuado *es un modelo con un elemento distinguido, e. j., dado un modelo $\mathcal{M} = \langle W, R, V\rangle$ y un elemento w en W, el modelo puntuado correspondiente es la tupla $\mathcal{M} = \langle w, W, R, V\rangle$. Si un modelo puntuado $\mathcal{M}$ satisface una fórmula φ escribimos $\mathcal{M} \models \varphi$.*

Ahora definimos la semántica para fórmulas en CNF modal.

Definición 2.6 (Semántica)**.** *Sea $\mathcal{M} = \langle w, W, R, V\rangle$ un modelo puntuado y φ una fórmula en CNF modal. Definimos inductivamente cuando una fórmula φ es* satisfecha *en $\mathcal{M}$ de la siguiente manera,*

$$
\begin{array}{lll}
\mathcal{M} \models \varphi & \textit{sii} & \textit{para todas las cláusulas } C \in \varphi,\ \mathcal{M} \models C,\\
\mathcal{M} \models C & \textit{sii} & \textit{existe un literal } l \in C \textit{ tal que } \mathcal{M} \models l,\\
\mathcal{M} \models p & \textit{sii} & p \in V(w) \ \textit{ para } \ p \in \mathsf{PROP},\\
\mathcal{M} \models \neg p & \textit{sii} & p \notin V(w) \ \textit{ para } \ p \in \mathsf{PROP},\\
\mathcal{M} \models \Box C & \textit{sii} & \langle v, W, R, V\rangle \models C, \textit{ para todo } v \textit{ tal que } wRv,\\
\mathcal{M} \models \neg\Box C & \textit{sii} & \mathcal{M} \not\models \Box C.
\end{array}
$$

Definición 2.7 (Consecuencia Semántica)**.** *Una fórmula φ es una* consecuencia semántica *de un conjunto de fórmulas Σ si para todo los modelos $\mathcal{M}$ y todos los puntos w en $\mathcal{M}$, si $\mathcal{M}, w \models \Sigma$ entonces $\mathcal{M}, w \models \varphi$, y en tal caso escribimos $\Sigma \models \varphi$. En vez de $\{\varphi\} \models \psi$ escribimos $\varphi \models \psi$.*

Dada una fórmula φ, $\mathsf{Mods}(\varphi) = \{\mathcal{M} \mid \mathcal{M} \models \varphi\}$ es la clase de todos los modelos de φ.

En las lógicas modales, el concepto básico para hablar de expresividad es el de *bisimulación* entre dos modelos.

La noción de bisimulación fue formulada por primera vez en [van Benthem, 1977; van Benthem, 1983], y establece que dos modelos relacionados por una bisimulación son *modalmente equivalentes*, es decir, satisfacen las mismas fórmulas.

Definición 2.8 (Bisimulación)**.** *Una bisimulación entre modelos $\mathcal{M} = \langle w, W, R, V\rangle$ y $\mathcal{M}' = \langle w', W', R', V'\rangle$ es una relación no vacía $Z \subseteq W \times W'$ que satisface las siguientes condiciones:*

i) wZw'. *[Raíz]*

ii) *Si wZw' entonces $p \in V(w)$ sii $p \in V'(w')$, para todo $p \in \mathsf{PROP}$.* *[Armonía Atómica]*

iii) *Si wZw' y wRv luego existe $v' \in W'$ tal que vZv' y $w'R'v'$.* *[Zig]*

iv) *Si wZw' y $w'R'v'$ luego existe $v \in W$ tal que vZv' y wRv.* *[Zag]*

Si existe una bisimulación entre modelos $\mathcal{M}$ y $\mathcal{M}'$ decimos que $\mathcal{M}$ y $\mathcal{M}'$ son bisimilares *y escribimos $\mathcal{M} \underline{\leftrightarrow} \mathcal{M}'$. Dos estados w y w' son bisimilares si están relacionados por una bisimulación, en cuyo caso escribimos $w \underline{\leftrightarrow} w'$.*

Una consecuencia importante de que dos modelos estén relacionados por una bisimulación es que ambos modelos satisfacen las mismas fórmulas, es decir, que la satisfacibilidad modal es *invariante bajo bisimulaciones* [Blackburn *et al.*, 2001].

Proposición 2.1 (Lema de Invarianza bajo Bisimulación). *Si $\mathcal{M} \leftrightarrows \mathcal{M}'$ luego, $\mathcal{M} \models \varphi$ si y solo si $\mathcal{M}' \models \varphi$, para toda φ.*

En lo que sigue trabajaremos con conjuntos de literales proposicionales, por lo cual necesitaremos las siguientes definiciones.

Definición 2.9 (Conjunto Completo, Consistente y Generado de literales proposicionales). *Un conjunto de literales proposicionales L es* completo *si por cada $p \in \mathsf{PROP}$, $p \in L$ o $\neg p \in L$. Es* consistente *si por cada $p \in \mathsf{PROP}$, $p \notin L$ o $\neg p \notin L$. Todo conjunto completo y consistente de literales proposicionales L define una única valuación proposicional $v \subseteq \mathsf{PROP}$ donde $p \in v$ si $p \in L$ y $p \notin v$ si $\neg p \in L$. Dado $S \subseteq \mathsf{PROP}$, el* conjunto consistente y completo de literales proposicionales generado por S *(notación L_S) es* $S \cup \{\neg p \mid p \in \mathsf{PROP} \backslash S\}$.

A continuación presentamos las definiciones básicas sobre teoría de grupos que necesitaremos más adelante (por más detalles sobre teoría de grupos ver [Fraleigh and Katz, 2003; Seress, 1997].

Definición 2.10 (Permutación). *Una* permutación de un conjunto A *es una biyección* $\sigma : A \mapsto A$.

El conjunto de todas las permutaciones sobre un conjunto A forma un grupo con la operación de composición de funciones como producto y la identidad como elemento neutral.

Teorema 2.1 (Grupo de Permutación). *Dado un conjunto no vacío A, sea S_A el conjunto de todas las permutaciones de A. Luego S_A es un grupo bajo composición de funciones.*

Dado $n \in \mathbb{Z}_{\geq 1}$ y una permutación σ, denotamos la composición de σ consigo misma n veces mediante σ^n. σ^0 denota la permutación identidad, σ^{-1} la permutación inversa de σ, y σ^{-n}, para $n \in \mathbb{Z}_{\geq 1}$ la composición de σ^{-1} consigo misma n veces.

Toda permutación σ de un conjunto A determina una partición de A en clases de equivalencia con la propiedad de que $a, b \in A$ están en la misma clase de equivalencia si y solo si $b = \sigma^n(a)$ para alguna $n \in \mathbb{Z}$.

Definición 2.11 (Órbitas de una permutación). *Dada una permutación σ de un conjunto A, las clases de equivalencia en A determinadas por σ son las* órbitas de σ.

Las permutaciones definidas sobre conjuntos finitos puede ser representadas utilizando *notación cíclica*.

Definición 2.12 (Notación Cíclica). *Sea A un conjunto finito y sean $a_1, \ldots, a_n$ elementos distintos de A. La expresión $(a_1\ a_2\ \ldots\ a_n)$ es un* ciclo *y denota la acción de mapear $a_1 \mapsto a_2, a_2 \mapsto a_3, \ldots, a_{n-1} \mapsto a_n, a_n \mapsto a_1$. Un elemento que no aparece en el ciclo se considera como dejado fijo por el ciclo. Un ciclo que contiene solo dos elementos se denomina una* transposición.

Dado que los ciclos definen permutaciones, pueden ser compuestos. Por lo tanto, toda permutación finita puede ser expresada como el producto de ciclos disjuntos [Fraleigh and Katz, 2003].

De ahora en más trabajaremos exclusivamente con permutaciones de literales proposicionales.

Definición 2.13 (Permutación de literales proposicionales)**.** *Una* permutación de literales proposicionales *es una función biyectiva* $\sigma : \mathsf{PLIT} \mapsto \mathsf{PLIT}$. *Dado un conjunto de literales proposicionales* L, $\sigma(L) = \{\sigma(l) \mid l \in L\}$.

Nota. La Definición 2.13 define permutaciones sobre el conjunto infinito PLIT. Sin embargo, en la práctica, solo trabajamos con permutaciones definidas sobre un subconjunto finito A de PLIT, es decir, el conjunto de los literales proposicionales que aparecen en la fórmula con la cual estamos trabajando. Por lo tanto, podemos utilizar todos los resultados concernientes a permutaciones sobre conjuntos finitos que se han introducido previamente. Podemos extender fácilmente una permutación sobre un subconjunto finito A de PLIT a una permutación sobre PLIT mapeando a si mismo todo elemento no en A.

En lo que sigue, asumiremos que toda permutación es una permutación de literales proposicionales y las llamaremos simplemente una "permutación".

Definición 2.14 (Permutación de una fórmula)**.** *Sea* φ *una fórmula en CNF modal y* σ *una permutación. Definimos* $\sigma(\varphi)$ *recursivamente como*

$$\begin{aligned} \sigma(\varphi) &= \{\sigma(C) \mid C \in \varphi\} && \text{para } \varphi \text{ una fórmula en CNF modal} \\ \sigma(C) &= \{\sigma(A) \mid A \in C\} && \text{para } C \text{ a una cláusula en CNF modal} \\ \sigma(\Box C) &= \Box\sigma(C) \\ \sigma(\neg\Box C) &= \neg\Box\sigma(C). \end{aligned}$$

Dado un conjunto de literales proposicionales, estamos interesados en aquellas permutaciones del conjunto que son *consistentes*.

Definición 2.15 (Permutación Consistente)**.** *Una permutación* σ *es* consistente *si por cada literal proposicional* l *tenemos que* $\sigma(\sim l) = \sim\sigma(l)$, *donde* $\sim$ *es una función que retorna el complemento de un literal proposicional, es decir,* $\sim p = \neg p$ *y* $\sim\neg p = p$.

La condición de consistencia de una permutación garantiza que la misma interactuará de forma correcta cuando es aplicada a un conjunto de literales proposicionales, es decir, si tenemos un conjunto consistente de literales proposicionales, se mantendrá consistente luego de aplicar una permutación consistente al mismo. Desde el punto de vista de la teoría de grupos, las permutaciones consistentes forman un subgrupo del grupo de todas las permutaciones sobre un determinado conjunto, ya que la composición de permutaciones consistentes da como resultado una permutación consistente.

Definición 2.16 (Simetría)**.** *Sea* φ *una fórmula en CNF modal. Una permutación consistente* σ *es una* simetría *de* φ *si* $\varphi = \sigma(\varphi)$, *cuando las conjunciones y disyunción en* φ *son representadas como conjuntos.*

Ahora presentamos el concepto clave para estudiar las simetrías: σ-simulaciones.

Definición 2.17 (σ-simulación)**.** *Sea* σ *una permutación. Una* σ-simulación *entre modelos* $\mathcal{M} = \langle w, W, R, V\rangle$ *y* $\mathcal{M}' = \langle w', W', R', V'\rangle$ *es una relación no vacía* $Z \subseteq W \times W'$ *que satisface las siguientes condiciones:*

i) wZw'. *[Raíz]*

ii) *Si wZw' luego $l \in L_{V(w)}$ si y solo si $\sigma(l) \in L_{V'(w')}$.* *[σ-Armonía]*

iii) *Si wZw' y wRv luego existe v' tal que $w'R'v'$ y vZv'.* *[Zig]*

iv) *Si wZw' y $w'R'v'$ luego existe v tal que wRv y vZv'.* *[Zag]*

Decimos que dos modelos puntuados $\mathcal{M}$ y $\mathcal{M}'$ son σ-similares (notación $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$) si existe una σ-simulación Z entre ellos.

Notar la semejanza existente con la definición de bisimulación (Definición 2.8). Una σ-simulación es, de hecho, una bisimulación que relaja la condición de armonía atómica para incorporar el efecto de las permutaciones. Sin embargo, en general, una σ-simulación no es una relación simétrica: podemos tener $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$ pero no $\mathcal{M}' \underline{\rightarrow}_\sigma \mathcal{M}$.

Si nos restringimos a permutaciones que puedan ser definidas como el producto de transposiciones disjuntas entonces una σ-simulación es una relación simétrica.

De la definición de σ-simulaciones se sigue que las mismas si bien no preservan la validez de las fórmulas modales (como es el caso de las bisimulaciones), preservan la validez de las *permutaciones* de fórmulas modales.

Proposición 2.2. *Sea σ una permutación consistente, φ una fórmula en CNF modal y $\mathcal{M} = \langle w, W, R, V\rangle$, $\mathcal{M}' = \langle w', W', R', V'\rangle$ modelos tales que $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$. Luego $\mathcal{M} \models \varphi$ sii $\mathcal{M}' \models \sigma(\varphi)$.*

Demostración. Ver [Orbe, 2014]. □

También necesitamos considerar el efecto de aplicar permutaciones a los modelos. Si φ es verdadera en un modelo $\mathcal{M}$, intuitivamente queremos que $\sigma(\varphi)$ sea verdadera en el modelo $\sigma(\mathcal{M})$.

Definición 2.18 (Permutación de un modelo)**.** *Sea σ una permutación y $\mathcal{M} = \langle w, W, R, V\rangle$ un modelo. Luego $\sigma(\mathcal{M}) = \langle w, W, R, V'\rangle$, donde,*

$$V'(w) = \sigma(L_{V(w)}) \cap \mathsf{PROP} \quad \textit{para todo } w \in W.$$

$L_{V(w)}$ es el conjunto de literales completo y consistente generado por$V(w)$. Dado M un conjunto de modelos, $\sigma(M) = \{\sigma(\mathcal{M}) \mid \mathcal{M} \in M\}$.

Una consecuencia de la definición anterior es que $\mathcal{M}$ y $\sigma(\mathcal{M})$ son siempre σ-similares.

Proposición 2.3. *Sea σ una permutación consistente y $\mathcal{M} = \langle w, W, R, V\rangle$ un modelo. Luego $\mathcal{M} \underline{\rightarrow}_\sigma \sigma(\mathcal{M})$.*

Demostración. Ver [Orbe, 2014]. □

Proposición 2.4. *Sea σ una permutación consistente, φ una fórmula en CNF modal y $\mathcal{M} = \langle w, W, R, V\rangle$ un modelo. Luego $\mathcal{M} \models \varphi$ si y solo si $\sigma(\mathcal{M}) \models \sigma(\varphi)$.*

Demostración. Se sigue directamente de las Proposiciones 2.2 y 2.3. □

Si σ es una simetría de φ luego, para todo modelo $\mathcal{M}$, $\mathcal{M}$ es un modelo de φ si y solo si $\sigma(\mathcal{M})$ también lo es. Esto es un corolario de las proposiciones anteriores en el caso particular que σ es una simetría de φ.

Corolario 2.1. *Si σ es una simetría de φ luego $\mathcal{M} \in Mods(\varphi)$ si y solo si $\sigma(\mathcal{M}) \in Mods(\varphi)$.*

El Corolario 2.1 nos indica que, en la lógica modal básica, las simetrías tienen el mismo efecto que en la lógica proposicional. El grupo de simetrías de una fórmula φ actuando sobre el conjunto de modelos lo particiona de forma tal que las clases de equivalencia (órbitas) contienen sólo modelos que satisfacen φ o sólo modelos que no satisfacen φ. Como consecuencia de esto, podríamos evitar buscar una solución en el espacio de modelos completos y enfocarnos solamente en los representantes de cada clase de equivalencia, suponiendo que los podemos computar.

Además de particionar el espacio de modelos, las simetrías nos proveen un mecanismo de inferencia.

Teorema 2.2. *Sean φ y ψ fórmulas en CNF modal y sea σ una simetría de φ. Luego $\varphi \models \psi$ si y solo si $\varphi \models \sigma(\psi)$.*

Demostración. Ver [Orbe, 2014]. □

El Teorema 2.2 nos provee de una mecanismo de inferencia que puede ser utilizado en toda situación donde exista algún tipo de consecuencia lógica involucrada durante el razonamiento modal automático. Sin lugar a dudas, aplicar una permutación es computacionalmente mas "barato" que aplicar una expansión en un tableaux o un paso de resolución. Por lo tanto, las nuevas fórmulas obtenidas de esta forma pueden reducir el tiempo total de ejecución de un algoritmo de inferencia. En el caso de la lógica proposicional este mecanismo de inferencia ha probado su efectividad en [Benhamou *et al.*, 2010]. En el caso modal todavía resta por determinar cuando este mecanismo puede ser utilizado. Estudiaremos como usar las simetrías de una fórmula en el capítulo 5.

El Corolario 2.1 y el Teorema 2.2 son los resultados claves que permiten utilizar las simetrías de una fórmula en la lógica modal básica.

2.2 MÁS ALLÁ DE LA LÓGICA MODAL BÁSICA

Hasta ahora hemos trabajado con simetrías en la lógica modal básica. A continuación generalizaremos los resultados obtenidos a lógicas modales más expresivas.

Para ello extenderemos nuestra teoría de simetrías al marco de los *modelos modales coinductivos*.

Los modelos modales coinductivos fueron introducidos en [Areces and Gorín, 2010] para investigar formas normales para diversas lógicas modales. Su principal característica es que permiten la representación de diferentes lógicas modales de forma homogénea. Los resultados obtenidos para los modelos modales coinductivos pueden ser extendidos fácilmente a lenguajes modales concretos seleccionando la clase de modelos apropiada.

A continuación presentamos la sintaxis y la semántica de los modelos modales coinductivos. Para más detalles sobre los mismos y ejemplos sobre como pueden definir diferentes lógicas modales referimos al lector a [Areces and Gorín, 2010].

2.2.1 *Modelos Modales Coinductivos*

Comencemos por definir el lenguaje modal que utilizaremos.

Definición 2.19 (Sintáxis). *Sea* $\mathsf{ATOM} = \{a_1, a_2, \ldots\}$ *un conjunto contable infinito de átomos y* $\mathsf{MOD} = \{m, m'', \ldots\}$ *un conjunto de símbolos de modalidades. El conjunto de las fórmulas modales* FORM *sobre la signatura* $\mathcal{S} = \langle \mathsf{ATOM}, \mathsf{MOD} \rangle$ *se define como*

$$\mathsf{FORM} ::= a \mid \neg\varphi \mid \varphi \vee \psi \mid [m]\varphi,$$

donde $a \in \mathsf{ATOM}$, $m \in \mathsf{MOD}$, *y* $\varphi, \psi \in \mathsf{FORM}$. $\top$ *y* $\bot$ *representan una tautología arbitraria y la contradicción, respectivamente. También utilizaremos los conectivos clásicos como* $\wedge, \rightarrow$ *definidos de la forma usual.*

Notar que el lenguaje definido es el mismo lenguaje que el definido para la lógica multi-modal básica en la Definición 2.1, sin embargo podremos modelar otras lógicas modales, como la lógica modal híbrida, utilizando el mismo lenguaje. Como lo hacemos, quedará claro cuando presentemos las definición de los modelos.

Por conveniencia trabajaremos con modelos puntuados (Definición 2.5).

Definición 2.20 (Modelos). *Sea* $\mathcal{S} = \langle \mathsf{ATOM}, \mathsf{MOD} \rangle$ *una signatura modal y* W *un conjunto no vacío.* Mods_W*, la clase de todos los modelos con dominio* W*, para la signatura* $\mathcal{S}$*, es el conjunto de tuplas* $\langle w, W, R, V \rangle$ *tales que:*

i) $w \in W$.

ii) $R(m, v) \subseteq \mathsf{Mods}_W$ *para* $m \in \mathsf{MOD}$ *y* $v \in W$.

iii) $V(v) \subseteq \mathsf{ATOM}$ *para* $v \in W$.

Mods *denota la clase de todos los modelos sobre todos los dominios, es decir,* $\mathsf{Mods} = \bigcup_W \mathsf{Mods}_W$.

Llamamos a w *el* punto de evaluación, *a* W *el* dominio, *a* V *la* valuación, *y a* R *la* relación de accesibilidad. *Para* $\mathcal{M}$ *un modelo arbitrario escribimos* $|\mathcal{M}|$ *para representar su dominio,* $w^{\mathcal{M}}$ *para representar su punto de evaluación,* $V^{\mathcal{M}}$ *para representar su valuación y* $R^{\mathcal{M}}$ *para representar su relación de accesibilidad. Con* $\mathsf{succs}^{\mathcal{M}}(m)$ *denotamos el conjunto* $R^{\mathcal{M}}(m, w^{\mathcal{M}})$ *de m-sucesores de* $w^{\mathcal{M}}$.

La principal diferencia entre la Definición 2.4 (de los modelos de Kripke) y la Definición 2.20 se encuentra en como definimos los m-sucesores. En la última, para cada modalidad m y cada estado w en el modelo, definimos $R(m, w)$, los sucesores de w a través de la modalidad m, como un conjunto de *modelos* (de ahí que la definición sea coinductiva), mientras que en la primera la definimos como un conjunto de puntos en el dominio.

En lo que sigue estamos interesados en trabajar con clases de modelos que sean *cerradas bajo relaciones de accesibilidad*.

Definición 2.21 (Extensión de un modelo). *Dado* $\mathcal{M} \in \mathsf{Mods}_W$*, sea* $\mathrm{Ext}(\mathcal{M})$*, la* extensión de $\mathcal{M}$*, el subconjunto más pequeño de* Mods_W *que contiene a* $\mathcal{M}$ *y es tal que si* $\mathcal{N} \in \mathrm{Ext}(\mathcal{M})$*, entonces* $R^{\mathcal{N}}(m, v) \subseteq \mathrm{Ext}(\mathcal{M})$ *para toda* $m \in \mathsf{MOD}$, $v \in W$.

Definición 2.22 (Clase Cerrada)**.** *Una clase no vacía de modelos $\mathcal{C}$ es* cerrada bajo relaciones de accesibilidad *(brevemente, la llamaremos* clase cerrada*) si siempre que $\mathcal{M} \in \mathcal{C}$ implica que* $\mathrm{Ext}(\mathcal{M}) \subseteq \mathcal{C}$.

Como en el caso de la lógica modal básica, trabajaremos con fórmulas en CNF modal.

Definición 2.23 (Literales y CNF Modal)**.** *Un* átomo literal *l es un átomo a o su negación $\neg a$. El conjunto de los literales sobre* ATOM *es* $\mathsf{ALIT} = \mathsf{ATOM} \cup \{\neg a \mid a \in \mathsf{ATOM}\}$. *Una fórmula modal está en* forma normal conjuntiva modal (en CNF modal) *si es una conjunción de cláusulas en CNF modal. Una* cláusula en CNF modal *es una disyunción de átomos literales y* literales modales*. Un* literal modal *es una fórmula de la forma $[m]C$ o $\neg[m]C$ donde C es una cláusula en CNF modal.*

Definición 2.24 (Semántica)**.** *Sea φ una fórmula en CNF modal y $\mathcal{M} = \langle w, W, R, V\rangle$ un modelo en Mods. Definimos $\models$ como*

$$\begin{array}{lll} \mathcal{M} \models a & sii & a \in V(w) \text{ para } a \in \mathsf{ATOM}, \\ \mathcal{M} \models \neg\varphi & sii & \mathcal{M} \not\models \varphi, \\ \mathcal{M} \models \varphi \vee \psi & sii & \mathcal{M} \models \varphi \text{ o } \mathcal{M} \models \psi, \\ \mathcal{M} \models [m]\varphi & sii & \mathcal{M}' \models \varphi, \text{ para todo } \mathcal{M}' \in R(m, w). \end{array}$$

Si $\mathcal{C}$ es una clase cerrada, escribimos $\mathcal{C} \models \varphi$ siempre que $\mathcal{M} \models \varphi$ para todo $\mathcal{M}$ en $\mathcal{C}$, y decimos que $\Gamma_{\mathcal{C}} = \{\varphi \mid \mathcal{C} \models \varphi\}$ es la lógica *definida por $\mathcal{C}$.*

Inspeccionando la definición anterior, vemos que la cláusula semántica que definimos para $[m]$ es la misma que define un operador box [Blackburn *et al.*, 2001], sin embargo, si nos restringimos a la clase de modelos apropiada, podemos asegurar que $[m]$ se comporta de diferentes formas, capturando la semántica de diferentes operadores modales.

2.2.2 *Una Teoría Generalizada de Simetrías*

Como en el caso de la lógica modal básica, trabajaremos con conjuntos de (átomos) literales completos y consistentes, y, como es de esperar, deberemos adaptar nuestra definición de permutación ya que ahora trabajamos con átomos literales.

Definición 2.25 (Permutación de átomos literales)**.** *Una* permutación *es una función biyectiva $\sigma : \mathsf{ALIT} \mapsto \mathsf{ALIT}$. Dado un conjunto de átomos literales L, $\sigma(L) = \{\sigma(l) \mid l \in L\}$.*

Dado que en nuestro lenguaje pueden ocurrir átomos dentro de las modalidades, como por ejemplo $@_i$, debemos tener esto en cuenta al momento de aplicar una permutación a una fórmula modal. Diremos que una modalidad es *indexada por átomos* si su definición depende del valor de un átomo. Si m es indexada por un átomo a escribimos $m(a)$.

Definición 2.26 (Permutación de una fórmula)**.** *Sea φ una fórmula en CNF modal y σ una permutación. Definimos $\sigma(\varphi)$ recursivamente como*

$$\begin{aligned}
\sigma(\varphi) &= \{\sigma(C) \mid C \in \varphi\} && \text{para } \varphi \text{ una fórmula en CNF modal} \\
\sigma(C) &= \{\sigma(A) \mid A \in C\} && \text{para } C \text{ una cláusula en CNF modal} \\
\sigma([m]C) &= [\sigma(m)]\sigma(C) \\
\sigma(\neg[m]C) &= \neg[\sigma(m)]\sigma(C)
\end{aligned}$$

donde $\sigma(m) = \sigma(m(a)) = m(\sigma(a))$ si m es indexada por a, y $\sigma(m) = m$ si no lo es.

Como en el caso de la lógica modal básica, nos restringiremos a trabajar con permutaciones consistentes.

Definición 2.27 (Permutaciones Consistentes y Simetrías)**.** *Una permutación σ es* consistente *si por cada literal l, $\sigma(\sim l) = \sim\sigma(l)$, donde $\sim$ es una función que retorna el complemento de un átomo literal, es decir, $\sim a = \neg a$ y $\sim\neg a = a$. Una permutación σ es una* simetría *de φ si $\varphi = \sigma(\varphi)$, cuando las conjunciones y disyunciones en φ son representadas como conjuntos.*

Ahora podemos generalizar los resultados obtenidos previamente al marco de los modelos modales coinductivos. Empecemos por la noción de σ-simulación.

Definición 2.28 (σ-simulación)**.** *Sea σ una permutación. Una* σ-simulación *entre modelos $\mathcal{M} = \langle w, W, R, V\rangle$ y $\mathcal{M}' = \langle w', W', R', V'\rangle$ es una relación no vacía $Z \subseteq \text{Ext}(\mathcal{M}) \times \text{Ext}(\mathcal{M}')$ que satisface las siguientes condiciones.*

i) $\mathcal{M} Z \mathcal{M}'$. *[Raíz]*

ii) $l \in L_{V(w)}$ *sii* $\sigma(l) \in L_{V'(w')}$. *[σ-Armonía]*

iii) Si $\mathcal{M} Z \mathcal{M}'$ *y* $\mathcal{N} \in R(m, w)$ *luego* $\mathcal{N} Z \mathcal{N}'$ *para algún* $\mathcal{N}' \in R'(\sigma(m), w')$. *[Zig]*

iv) Si $\mathcal{M} Z \mathcal{M}'$ *y* $\mathcal{N}' \in R'(m, w')$ *luego* $\mathcal{N} Z \mathcal{N}'$ *para algún* $\mathcal{N} \in R(\sigma^{-1}(m), w)$. *[Zag]*

Decimos que dos modelos $\mathcal{M}$ y $\mathcal{M}'$ son σ-similares (notación $\mathcal{M} \rightarrow_\sigma \mathcal{M}'$) si existe una σ-simulación Z entre ellos.

Notar que la definición de σ-simulación tiene en cuenta el hecho de que existen modalidades indexadas por átomos al considerar la permutación cuando se accede a los sucesores, e. j., $R'(\sigma(m), w')$.

Como en el caso de la lógica modal básica, una σ-simulación preserva la validez de las permutaciones de fórmulas.

Proposición 2.5. *Sea σ una permutación consistente, φ una fórmula en CNF modal y $\mathcal{M} = \langle w, W, R, V\rangle$, $\mathcal{M}' = \langle w', W', R', V'\rangle$ modelos tales que $\mathcal{M} \rightarrow_\sigma \mathcal{M}'$. Luego $\mathcal{M} \models \varphi$ si y solo si $\mathcal{M}' \models \sigma(\varphi)$.*

Demostración. Ver [Orbe, 2014]. □

Ahora definimos como aplicar permutaciones a los modelos modales coinductivos.

Definición 2.29 (Permutación de un modelo). *Sea σ una permutación y $\mathcal{M} = \langle w, W, R, V\rangle$ un modelo. Luego $\sigma(\mathcal{M}) = \langle w, W, R', V'\rangle$, donde,*

$$\begin{aligned} V'(v) &= \sigma(L_{V(v)}) \cap \mathsf{ATOM} \quad \textit{para todo } v \in W, \textit{ y,} \\ R'(m, v) &= \{\sigma(\mathcal{N}) \mid \mathcal{N} \in R(\sigma(m), v)\} \textit{ para toda } m \in \mathsf{MOD} \textit{ y } v \in W. \end{aligned}$$

Dado un conjunto de modelos M, $\sigma(M) = \{\sigma(\mathcal{M}) \mid \mathcal{M} \in M\}$.

Además de modificar la valuación, aplicar una permutación a un modelo modal coinductivo implica propagar la permutación a todos los modelos accesibles.

Se sigue de la definición anterior que $\mathcal{M}$ y $\sigma(\mathcal{M})$ son σ-similares.

Proposición 2.6. *Sea σ una permutación consistente y $\mathcal{M} = \langle w, W, V, R\rangle$ un modelo. Luego $\mathcal{M} \rightarrow_\sigma \sigma(\mathcal{M})$.*

Demostración. Ver [Orbe, 2014]. □

Proposición 2.7. *Sea σ una permutación consistente, $\mathcal{M}$ un modelo y φ una fórmula en CNF modal. Luego $\mathcal{M} \models \varphi$ si y solo si $\sigma(\mathcal{M}) \models \sigma(\varphi)$.*

Demostración. Se sigue de las Proposiciones 2.6 y 2.5. □

Los resultados previos nos llevan al corolario deseado.

Corolario 2.2. *Si σ es una simetría de φ luego $\mathcal{M} \in \mathit{Mods}(\varphi)$ si y solo si $\sigma(\mathcal{M}) \in \mathit{Mods}(\varphi)$.*

El Corolario 2.2 nos dice que las simetrías en el marco coinductivo tienen el mismo efecto que para la lógica modal básica: particionan el espacio de modelos en clases de equivalencia de forma tal que cada clase de equivalencia contiene solo modelos que satisfacen la fórmula o solo modelos que no la satisfacen. La importancia de este corolario radica en que, ya que se ha probado en el marco coinductivo, lo podemos aplicar a toda lógica modal que se pueda representar dentro de este marco.

Notar, que la noción de σ-simulación en el marco de los modelos modales coinductivos es lo suficientemente general para ser aplicada a un gran rango de lógicas modales. Sin embargo, la definición de σ-simulación no asume nada sobre los si los modelos en cuestión están dentro de la misma clase de modelos. Sin embargo, cuando trabajamos con simetrías de una fórmula esto no es un problema, y podemos estar seguros de que ambos modelos están dentro de la misma clase. Esto se debe a que una simetría de una fórmula es, implícitamente, una permutación *tipada*: solo mapea símbolos de forma tal de que la fórmula resultante es una fórmula del lenguaje. Si este no es el caso, o la fórmula no está en el lenguaje de la lógica o la permutación no es una simetría. Por lo tanto, podemos pensar que la definición del lenguaje restringe los mapeos posibles.

Ahora podemos probar que las simetrías pueden ser utilizadas como mecanismo de inferencia en el marco coinductivo.

Teorema 2.3. *Sean φ y ψ fórmulas modales, sea σ una simetría de φ. Luego $\varphi \models \psi$ si y solo si $\varphi \models \sigma(\psi)$.*

Demostración. Ver [Orbe, 2014]. □

2.2.3 *Permutaciones en Capas*

Ahora presentamos la noción de *permutaciones en capas (layered permutations)* que, en casos donde el lenguaje modal tiene la propiedad de modelo arbolar, nos permite desarrollar una noción más flexible de simetría, donde la idea principal es utilizar diferentes permutaciones a cada profundidad modal. De esta forma, podemos encontrar simetrías que no serían encontradas de otra forma.

Comenzamos por definir la propiedad de modelo arbolar (tree model property) [Blackburn *et al.*, 2001] para modelos modales coinductivos.

Definición 2.30 (Caminos en un modelo)**.** *Dado un modelo $\mathcal{M}$, un* camino (finito) con raíz en $\mathcal{M}$ *es una secuencia $\pi = (\mathcal{M}_0, m_1, \mathcal{M}_1, \ldots, m_k, \mathcal{M}_k)$, para $m_i \in \mathsf{MOD}$ donde $\mathcal{M}_0 = \mathcal{M}$, $k \geq 0$, y $\mathcal{M}_i \in R(m_i, w^{\mathcal{M}_{i-1}})$ para $i = 1, \ldots, k$. Para un camino $\pi = (\mathcal{M}_0, m_1, \mathcal{M}_1, \ldots, m_k, \mathcal{M}_k)$ definimos $first(\pi) = \mathcal{M}_0$, $last(\pi) = \mathcal{M}_k$, y $length(\pi) = k$. Para un camino $\pi = (\mathcal{M}_0, m_1, \mathcal{M}_1, \ldots, m_k, \mathcal{M}_k)$, un modelo $\mathcal{N}$ y una modalidad $m \in \mathsf{MOD}$, tal que $\mathcal{N} \in R(m, w^{\mathcal{M}_k})$, con $\pi m\mathcal{N} = (\mathcal{M}_0, m_1, \mathcal{M}_1, \ldots, m_k, \mathcal{M}_k, m, \mathcal{N})$ denotamos la extensión de π por $\mathcal{N}$ a través de m. Denotamos con $\Pi[\mathcal{M}]$ al conjunto de todos los caminos con raíz en $\mathcal{M}$.*

Un *modelo arbolar coinductivo* es una modelo que tiene un único camino hacia todo modelo alcanzable (todo modelo en $\mathrm{Ext}(\mathcal{M})$). Formalmente podemos definir la clase de los modelos arbolares coinductivos, $\mathcal{C}_{Tree}$, con la siguiente condición:

$$\mathcal{C}_{Tree} := P_{Tree}(\mathcal{M}) \iff last : \Pi[\mathcal{M}] \mapsto \mathrm{Ext}(\mathcal{M}) \text{ es biyectiva.}$$

Por ejemplo, la construcción de *desenredado* (*unraveling*), en su versión para modelos modales coinductivos, siempre define un modelo en $\mathcal{C}_{Tree}$.

Definición 2.31 (Desenredado de un Modelo)**.** *Dado un modelo $\mathcal{M} = \langle w, W, R, V \rangle$, el* desenredado *de $\mathcal{M}$, (notación $\mathcal{T}(\mathcal{M})$), es el modelo modal coinductivo con raíz $\mathcal{T}(\mathcal{M}) = \langle (\mathcal{M}), \Pi[\mathcal{M}], R', V' \rangle$ donde*

$$\begin{aligned} V'(\pi) &= V(w^{last(\pi)}), \text{ para todo } \pi \in \Pi[\mathcal{M}], \\ R'(m, \pi) &= \{\langle \pi', \Pi[\mathcal{M}], R', V' \rangle \mid \text{existe } \mathcal{N} \in \mathsf{Mods}_W \text{ t.q. } \pi m\mathcal{N} = \pi', \pi \neq \pi'\}, \end{aligned}$$

para $m \in \mathsf{MOD}, \pi \in \Pi[\mathcal{M}]$.

Es fácil verificar que, dado un modelo $\mathcal{M}$, el modelo desenredado $\mathcal{T}(\mathcal{M})$ es un árbol ($\mathcal{T}(\mathcal{M}) \in \mathcal{C}_{Tree}$) y, como es de esperar, $\mathcal{M}$ y $\mathcal{T}(\mathcal{M})$ son bisimilares.

A continuación, utilizaremos los árboles para definir un tipo más flexible de simetrías que llamaremos *simetrías en capas* (*layered symmetries*). La siguiente propiedad nos da una condición suficiente que asegura que las simetrías en capas preservan la consecuencia lógica.

Definición 2.32 (Propiedad de clausura de modelos arbolares)**.** *Decimos que una clase de modelos $\mathcal{C}$ está* cerrada bajo arboles *si por cada modelo $\mathcal{M} \in \mathcal{C}$ existe un modelo arbolar $\mathcal{T} \in \mathcal{C}$ tal que $\mathcal{M} \underline{\leftrightarrow} \mathcal{T}$.*

De la Definición 2.32 se sigue que una clase de modelos $\mathcal{C}$ cerrada bajo desenredamientos ($\mathcal{T}(\mathcal{M}) \in \mathcal{C}$ para todo $\mathcal{M} \in \mathcal{C}$) es también cerrada bajo arboles.

Las lógicas modales definidas sobre clases de modelos cerradas bajo arboles poseen una propiedad interesante: existe una correlación directa entre la profundidad modal sintáctica de una fórmula y la profundidad en un modelo arbolar que la satisface. En los modelos arbolares, una noción de capa es inducida por la profundidad (distancia desde la raíz) de los nodos en el modelo. A su vez, en las fórmulas modales una noción de capa es inducida por el anidamiento de los operadores modales. Una consecuencia de esta correspondencia es que los literales que ocurren en diferentes capas de una fórmula son semánticamente diferentes [Areces *et al.*, 2000], es decir, un literal puede ser asignado diferentes valores en las diferentes capas.

Esta independencia de los literales en las distintas capas nos permite definir una noción más flexible de permutación que llamamos *permutación en capas*. Clave para esta noción de permutación en capas es la noción de *secuencia de permutaciones*.

Definición 2.33 (Permutación en Capas)**.** *Definimos una* permutación en capas $\bar{\sigma}$ *como* $\bar{\sigma} = \langle\rangle$ *(es decir,* $\bar{\sigma}$ *es la secuencia vacía) o* $\bar{\sigma} = \sigma : \bar{\sigma}_2$ *con* σ *una permutación y* $\bar{\sigma}_2$ *una secuencia de permutaciones. Alternativamente podemos usar la notación* $\bar{\sigma} = \langle\sigma_1, \ldots, \sigma_n\rangle$ *en vez de* $\bar{\sigma} = \sigma_1{:}\ldots{:}\sigma_n{:}\langle\rangle$.

Sea $|\bar{\sigma}| = n$ *la longitud de* $\bar{\sigma}$ *(*$\langle\rangle$ *tiene longitud* 0*). Para* $1 \leq i \leq n$*, escribimos* $\bar{\sigma}_i$ *para la subsecuencia que comienza en i-ésimo elemento de* $\bar{\sigma}$*. Para* $i \geq n$*, definimos* $\bar{\sigma}_i = \langle\rangle$*. En particular* $\bar{\sigma} = \bar{\sigma}_1$*. Dada una permutación en capas* $\sigma_1 : \bar{\sigma}_2$ *definimos* $head(\sigma_1 : \bar{\sigma}_2) = \sigma_1$ *y* $head(\langle\rangle) = \sigma_{Id}$*, donde* σ_{Id} *es la permutación identidad. Decimos que una permutación en capas es* consistente *si todas sus permutaciones son consistentes.*

Definición 2.34 (Permutación en capas de una fórmula)**.** *Sea* φ *una fórmula en CNF modal y* $\bar{\sigma}$ *una permutación en capas. Definimos* $\bar{\sigma}(\varphi)$ *recursivamente como*

$$\begin{array}{rcll} \langle\rangle(\varphi) & = & \varphi & \\ (\sigma_1 : \bar{\sigma}_2)(l) & = & \sigma_1(l) & \text{para } l \in \mathsf{ALIT} \\ (\sigma_1 : \bar{\sigma}_2)([m]C) & = & [\sigma_1(m)]\bar{\sigma}_2(C) & \\ \bar{\sigma}(C) & = & \{\bar{\sigma}(A) \mid A \in C\} & \text{para } C \text{ una cláusula o una fórmula.} \end{array}$$

Notar que las permutaciones en capas están bien definidas aún si la profundidad modal de la fórmula es mayor que la longitud de la permutación en capas. Las permutaciones en capas nos permiten definir una permutación distinta a cada profundidad modal. Esto nos permite encontrar simetrías que no encontraríamos de otra forma.

De ahora en más podemos repetir el trabajo realizado en las secciones anteriores para arribar a un resultado similar al Teorema 2.3 pero ahora sobre permutaciones en capas, con una sola observación: la extensión de la noción de permutación de un modelo $\sigma(\mathcal{M})$ a permutaciones en capas solo puede ser realizada si el modelo $\mathcal{M}$ es un modelo arbolar. Esto implica que debemos considerar el requerimiento adicional de que la clase de modelos $\mathcal{C}$ sea cerrada bajo arboles para que el resultado sea válido.

Definición 2.35 (Permutación en capas de un modelo)**.** *Sea* $\bar{\sigma}$ *una permutación en capas y* $\mathcal{M} = \langle w, W, R, V\rangle$ *un modelo arbolar. Luego* $\bar{\sigma}(\mathcal{M}) = \langle w, W, R', V'\rangle$*, donde,*

$$\begin{array}{rcll} V'(v) & = & head(\bar{\sigma})(L_{V(v)}) \cap \mathsf{ATOM} & \text{para todo } v \in W, \text{ y,} \\ R'(m, v) & = & \{\bar{\sigma}_2(\mathcal{N}) \mid \mathcal{N} \in R(head(\bar{\sigma})(m), v)\} & \text{para todo } m \in \mathsf{MOD} \text{ y } v \in W. \end{array}$$

Para M un conjunto de modelos arbolares, $\bar{\sigma}(M) = \{\bar{\sigma}(\mathcal{M}) \mid \mathcal{M} \in M\}$.

Ahora podemos extender la noción de σ-simulación a permutaciones en capas.

Definición 2.36 ($\bar{\sigma}$-simulación)**.** *Sea $\bar{\sigma}$ una permutación en capas. Una $\bar{\sigma}$-*simulación *entre modelos $\mathcal{M} = \langle w, W, R, V\rangle$ y $\mathcal{M}' = \langle w', W', R', V'\rangle$ es una familia de relaciones $Z_{\bar{\sigma}_i} \subseteq \mathrm{Ext}(\mathcal{M}) \times \mathrm{Ext}(\mathcal{M}')$, $1 \leq i$, que satisface las siguientes condiciones:*

i) $\mathcal{M} Z_{\bar{\sigma}_1} \mathcal{M}'$. *[Raíz]*

ii) Si $w Z_{\bar{\sigma}_i} w'$ *luego* $l \in L_{V(w)}$ *sii* $head(\bar{\sigma}_i)(l) \in L_{V'(w')}$. *[$\bar{\sigma}$-Armonía]*

iii) Si $\mathcal{M} Z_{\bar{\sigma}_i} \mathcal{M}'$ *y* $\mathcal{N} \in R(m, w)$ *luego* $\mathcal{N} Z_{\bar{\sigma}_{i+1}} \mathcal{N}'$ *para algún* $\mathcal{N}' \in R'(head(\bar{\sigma}_i)(m), w')$. *[Zig]*

iv) Si $\mathcal{M} Z_{\bar{\sigma}_i} \mathcal{M}'$ *y* $\mathcal{N}' \in R'(m, w')$ *luego* $\mathcal{N} Z_{\bar{\sigma}_{i+1}} \mathcal{N}'$ *para algún* $\mathcal{N} \in R(head(\bar{\sigma}_i)^{-1}(m), w)$. *[Zag]*

Decimos que dos modelos $\mathcal{M}$ y $\mathcal{M}'$ son $\bar{\sigma}$-similares (notación $\mathcal{M} \rightleftarrows_{\bar{\sigma}} \mathcal{M}'$), si hay una $\bar{\sigma}$-simulación entre ellos.

Notar que la definición anterior no hace ninguna suposición sobre el tamaño de la permutación en capas. De hecho, esta bien definida aún si la misma es la secuencia vacía. En ese caso, se comporta como la permutación identidad a cada profundidad modal, por lo cual la relación define una bisimulación entre modelos.

Ahora podemos probar el resultado principal que vincula las simetrías en capas y la consecuencia lógica.

Teorema 2.4. *Sean φ y ψ fórmulas en CNF modal, sea $\bar{\sigma}$ una permutación en capas consistente, y sea $\mathcal{C}$ una clase de modelos cerrada baja arboles. Si $\bar{\sigma}$ es una simetría de φ luego para toda ψ tenemos que $\varphi \models_{\mathcal{C}} \psi$ si y solo si $\varphi \models_{\mathcal{C}} \bar{\sigma}(\psi)$.*

Demostración. Ver [Orbe, 2014]. □

Parte II

DETECTANDO Y UTILIZANDO SIMETRÍAS

"Socrates: *Then, if we are not able to hunt the goose with one idea, with three we may take our prey; Beauty, Symmetry, Truth are the three. . .*"

Plato. *Philebus*. 65a.

3

DETECCIÓN DE SIMETRÍAS PARA LÓGICAS MODALES

El primer paso para utilizar las simetrías en cualquier lógica es detectarlas. Mucho trabajo ha sido realizado con este fin, resultando en diferentes técnicas de detección. En este capítulo nos enfocaremos en detectar simetrías en fórmulas en CNF modal usando técnicas basadas en grafos.

La detección de simetrías basada en grafos es la técnica más común para detectar simetrías en fórmulas y ha encontrado aplicación en diferentes dominios [Sakallah, 2009].

Su éxito se puede explicar en función de dos aspectos. Primero, es conceptualmente simple: la idea es construir un grafo a partir de una fórmula, de forma tal, que el grupo de automorfismos del grafo sea isomorfo al grupo de simetrías de la fórmula. Segundo, la disponibilidad de herramientas de detección de automorfismos de grafos muy eficientes, como ser `Nauty` [McKay, 2007], `Saucy` [Darga *et al.*, 2004; Darga *et al.*, 2008; Katebi *et al.*, 2012] y `Bliss` [Junttila and Kaski, 2007], y que, además, pueden manejar grafos de gran tamaño, facilita la implementación e integración de este tipo de detección en los solvers actuales.

En este capítulo primero introducimos las definiciones y notación necesarias (Sección 3.1). Luego presentamos un algoritmo para detectar *simetrías globales*, en fórmulas en CNF modal, para un amplio abanico de lógicas modales (Sección 3.2). Luego extendemos el algoritmo para detectar *simetrías en capas* en aquellas lógicas que tienen la propiedad de modelo arbolar (Sección 3.3). Finalmente, implementamos los algoritmos para la lógica modal básica y lo evaluamos sobre distintos conjuntos de fórmulas modales (Sección 3.4).

Los resultados presentados en este capítulo fueron publicados en [Areces *et al.*, 2012; Areces and Orbe, 2013].

3.1 DEFINICIONES

A continuación trabajaremos en el contexto de los modelos modales coinductivos definidos en la Sección 2.2.

A menos que se indique lo contrario, trabajaremos con fórmulas en CNF modal, considerándolas como conjuntos de conjuntos (ver Sección 2.1), aunque las escribiremos de la forma usual para mantener la claridad en la notación. Con $Sub(\varphi)$ denotaremos al conjunto de subfórmulas de φ.

Definición 3.1 (Átomos de una fórmula)**.** *Sea φ una fórmula en CNF modal. Con $At(\varphi)$ denotamos al conjunto de átomos que ocurren en φ independientemente de la profundidad modal a la cual ocurren. Con $At(\varphi, n)$, $n \in \mathbb{N}$, denotamos al conjunto de átomos que ocurren en φ a profundidad modal n.*

Definición 3.2 (Cláusulas top y cláusulas modales)**.** *Sea φ una fórmula en CNF modal. Una* cláusula top *de φ es una cláusula que ocurre a profundidad modal 0. Una* cláusula modal *de φ es una cláusula que ocurre en un literal modal.*

Un aspecto clave en la construcción de un grafo coloreado es como colorear los nodos. Para ello definimos una *función de tipado*

Definición 3.3 (Función de Tipado). *Sea* $s : \mathsf{MOD} \times \{0,1\} \mapsto \mathbb{N}\backslash\{0,1\}$ *una función inyectiva y sea* $t : Sub(\varphi) \mapsto \mathbb{N}$ *una función parcial definida como:*

$$t(\psi) = \begin{cases} 1 & \text{si } \psi \text{ es una cláusula top .} \\ s(m,0) & \text{si } \psi = [m]C. \\ s(m,1) & \text{si } \psi = \neg[m]C. \end{cases}$$

La función de tipado t *asigna un tipo numérico a toda cláusula (top o modal). Para las cláusulas modales, el tipo esta basado en la modalidad y la polaridad del literal modal en la cual ocurre.*

De ahora en más, llamaremos *permutación (simetría) global* a una permutación (simetría) como las definidas en la Definición 2.25, y *permutación (simetría) en capas* a una permutación como las definidas en la Definición 2.33.

3.2 DETECTANDO SIMETRÍAS GLOBALES

Ahora presentamos un algoritmo para detectar *simetrías globales* en fórmulas en CNF modal. El algoritmo está basado en el algoritmo MIN3C definido para fórmulas proposicionales en [Aloul *et al.*, 2003b].

Al igual que el algoritmo MIN3C, nuestro algoritmo construye un grafo coloreado a partir de una fórmula en CNF modal, de forma tal que el grupo de automorfismos del grafo es isomorfo al grupo de simetrías de la fórmula.

Definición 3.4 (Algoritmo para Simetrías Globales). *Sea* φ *una fórmula en CNF modal y* t *una función de tipado. El grafo* $G(\varphi) = (V, E_1, E_2)$ *se construye de la siguiente forma:*

i) Para cada átomo $a \in At(\varphi)$*:*

  a) *Agregar dos nodos de color (tipo)* 0*: uno para el literal positivo* a *y otro para el literal negativo* $\neg a$*.*

  b) *Agregar un arco* E_1 *entre el literal positivo y el literal negativo para asegurar consistencia Booleana.*

ii) Para cada cláusula top C *en* φ*:*

  a) *Agregar un nodo* cláusula *de color* $t(C)$*.*

  b) *Para cada átomo literal* l *que ocurre en* C*, agregar un arco* E_1 *entre el nodo para* C *y el nodo para* l*.*

  c) *Para cada literal modal* $[m]C'$ *(*$\neg[m]C'$*) que ocurre en* C*:*

    1) *Agregar un nodo cláusula de color* $t([m]C')$ *(*$t(\neg[m]C')$*) para representar la cláusula modal* C'*.*

    2) *Agregar un arco* E_1 *entre el nodo para* C *y el nodo para* C'*.*

    3) *Si* m *está indexada por un átomo literal* l *agregar un arco* E_2 *entre el nodo para* C' *y el nodo para el literal* l*.*

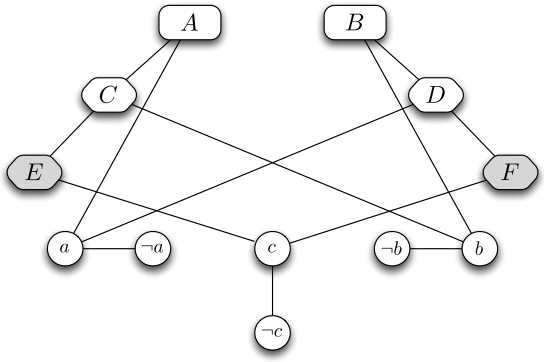
4) *Repetir el proceso desde punto ii)*b *por cada literal (átomo o modal) que ocurre en* C'.

Para una fórmula con A *átomos,* C *cláusulas top,* M *cláusulas modales, y* R *modalidades, ésta construcción produce un grafo con* $2 + 2R$ *colores y* $(2|A| + C + M)$ *nodos.*

Ejemplo 3.1. *Considere la fórmula*

$$\varphi = (a \vee [m](b \vee \neg[m]c)) \wedge (b \vee [m](a \vee \neg[m]c)).$$

La Figura 3.1 muestra su grafo $G(\varphi)$ *asociado construido usando el algoritmo de la Definición 3.4 (los colores están representados por las diferentes formas en la figura).*



$A = (a \vee [m](b \vee \neg[m]c))$
$B = (b \vee [m](a \vee \neg[m]c))$
$C = [m](b \vee \neg[m]c)$
$D = [m](a \vee \neg[m]c)$
$E = \neg[m]c$
$F = \neg[m]c$

Figura 3.1: Ejemplo del algoritmo para simetrías globales.

El grafo tiene un automorfismo no trivial $\pi = (A\ B)(C\ D)(E\ F)(a\ b)(\neg a\ \neg b)$ *que se corresponde con la simetría* $\sigma = (a\ b)(\neg a\ \neg b)$ *de* φ.

El siguiente resultado prueba que la construcción definida en la Definición 3.4 es correcta.

Teorema 3.1. *Sea* φ *una fórmula en CNF modal y* $G(\varphi) = (V, E_1, E_2)$ *el grafo coloreado construido a partir de la Definición 3.4. Luego, toda simetría* σ *de* φ *se corresponde uno-a-uno con un automorfismo* π *de* $G(\varphi)$.

Demostración. Ver [Orbe, 2014]. □

El Teorema 3.1 nos asegura que el algoritmo es correcto y por ende, que no se detectarán simetrías espúreas, es decir, automorfismos del grafo que no son simetrías de la fórmula.

Dado que esta construcción esta desarrollada en el marco coinductivo y que la misma no hace ninguna suposición sobre la clase de modelos sobre la cual se interpretan las fórmulas, podemos pensarla como un algoritmo *modelo* a partir de la cual derivar algoritmos para lógicas modales concretas. Para algunas lógicas modales, como la lógica modal básica, la derivación es directa, y podemos utilizar el algoritmo tal como lo hemos definido. Para otras lógicas, sin embargo, la derivación debe ser realizada con más cuidado. Por ejemplo, para lógicas modales con modalidades indexadas por átomos, como las lógicas híbridas, implementaciones prácticas

del algoritmo requieren necesariamente una modificación del mismo ya que las herramientas actuales de detección de automorfismos de grafo no pueden manejar distintos tipos de arcos. En este caso, sin embargo, la modificación es sencilla, ya que podemos reemplazar los arcos E_2 por un nodo adicional (un nodo de *indexado*) y dos arcos: uno entre el nodo del literal que indexa y el nodo de indexado y otro entre éste nodo y la modalidad indexada.

3.3 DETECTANDO SIMETRÍAS EN CAPAS

Ahora presentamos una extensión al algoritmo presentado en la Definición 3.4 que permite la detección de simetrías en capas para lógicas modales que tienen la propiedad de modelo arbolar. La observación clave es, que en estas lógicas, los literales que ocurren a diferentes profundidades modales son semánticamente diferentes y, por lo tanto, pueden ser considerados independientemente. Por lo tanto, podemos definir permutaciones en capas que tengan diferentes permutaciones a cada profundidad modal (ver Definición 2.33).

Definición 3.5 (Algoritmo para Simetrías en Capas)**.** *Sea φ una fórmula en CNF modal con profundidad modal n y t una función de tipado. El grafo $G(\varphi) = (V, E_1, E_2)$ se construye de la siguiente forma:*

i) Por cada átomo $a \in At(\varphi, i)$ con $0 \leq i \leq n$:

  a) *Agregar dos nodos de color 0: uno para el literal positivo y otro para el literal negativo. Etiquetar los nodos como a_i y $\neg a_i$ respectivamente (el subíndice i indica la profundidad modal del literal).*

  b) *Agregar un arco E_1 entre el nodo del literal positivo y el nodo del literal negativo para asegurar consistencia Booleana.*

ii) Por cada cláusula top C en φ:

  a) *Agregar un nodo* cláusula *de color $t(C)$.*

  b) *Por cada literal átomo l que ocurre en C, agregar un arco E_1 entre el nodo de C y el nodo de $l_{md(C)}$, donde $md(C)$ es la profundidad modal a la cual ocurre C.*

  c) *Por cada literal modal $[m]C'$ ($\neg[m]C'$) que ocurre en C:*

    1) *Agregar un nodo cláusula de color $t([m]C')$ ($t(\neg[m]C')$) para representar la cláusula modal C'.*

    2) *Agregar un arco E_1 entre el nodo de C y el nodo de C'.*

    3) *Si m es indexada por un literal átomo l, agregar un arco E_2 entre el nodo de C' y el nodo del literal $l_{md(C')}$.*

    4) *Repetir el proceso desde el punto ii)b para cada literal (átomo o modal) que ocurre en C'.*

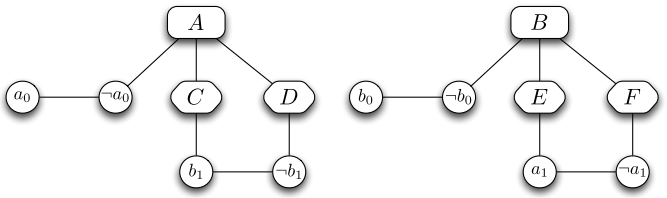
Dada una fórmula de profundidad modal n con A átomos, C cláusulas top, M cláusulas modales, y R modalidades, ésta construcción produce un grafo con $2 + 2R$ colores y a lo sumo $(2|A| \times (n + 1) + C + M)$ nodos.

Este algoritmo difiere del presentado en la Definición 3.4 en la forma en la cual los átomos literales son manejados durante la construcción de $G(\varphi)$: si un literal ocurre a diferentes profundidades modales, el algoritmo trata cada una de estas ocurrencias independientemente, agregando distintos nodos. De esta forma se incorpora la noción de capas que se introdujo en el Capítulo 2.2.3.

Ejemplo 3.2. *Considere la fórmula*

$$\varphi = (\neg a \vee [m]b \vee [m]\neg b) \wedge (\neg b \vee [m]a \vee [m]\neg a).$$

La Figura 3.2 muestra su grafo coloreado asociado $G(\varphi)$, construido usando el algoritmo definido en la Definición 3.5.



$$A = (\neg a \vee [m]b \vee [m]\neg b)$$
$$B = (\neg b \vee [m]a \vee [m]\neg a)$$
$$C = [m]b$$
$$D = [m]\neg b$$
$$E = [m]a$$
$$F = [m]\neg a$$

Figura 3.2: Ejemplo del algoritmo para simetrías en capas.

El grupo de automorfismos del grafo es generado por los siguientes tres generadores:

$$\text{Gens. } Aut(G(\varphi)): \quad \begin{aligned} \pi_1 &= (C\ D)(b_1\ \neg b_1) \\ \pi_2 &= (E\ F)(a_1\ \neg a_1) \\ \pi_3 &= (A\ B)(C\ E)(D\ F)(a_0\ b_0)(\neg a_0\ \neg b_0)(a_1\ b_1)(\neg a_1\ \neg b_1). \end{aligned}$$

Estos automorfismos se corresponden con las siguientes simetrías, las cuales son generadores del grupo de simetrías de la fórmula φ (σ_{Id} es la permutación identidad):

$$\text{Gens. } Sym(\varphi): \quad \begin{aligned} \bar{\sigma}_1 &= \langle \sigma_{Id}, (b\ \neg b) \rangle \\ \bar{\sigma}_2 &= \langle \sigma_{Id}, (a\ \neg a) \rangle \\ \bar{\sigma}_3 &= \langle (a\ b)(\neg a\ \neg b), (a\ b)(\neg a\ \neg b) \rangle. \end{aligned}$$

Proposición 3.1. *Sea φ una fórmula en CNF modal y $G(\varphi) = (V, E_1, E_2)$ el grafo coloreado construido a partir de la Definición 3.5. Luego, toda simetría $\bar{\sigma}$ de φ se corresponde uno-a-uno con un automorfismo π de $G(\varphi)$.*

Demostración. Ver [Orbe, 2014]. □

3.4 EVALUACIÓN EXPERIMENTAL

En esta sección implementamos los algoritmos de detección para la lógica modal básica, y los probamos en diferentes conjuntos de fórmulas (benchmarks) modales. Por simplicidad, nos restringimos al caso mono-modal, ya que la generalización al caso multi-modal es directa. Para más información y referencias sobre las metodologías de evaluación experimental en lógicas modales ver [Horrocks *et al.*, 2000].

3.4.1 *Implementación*

Implementamos ambos algoritmos en la herramienta `sy4ncl`[1]. `sy4ncl` es una herramienta de línea de comando, desarrollada en Haskell, que toma una fórmula de la lógica modal básica en formato `intohylo`, construye el grafo seleccionado (global o en capas) y lo disponibiliza en formato `Bliss` o `Saucy`, junto con un mapeo entre nodos y literales y estadísticas sobre el grafo.

Por cada fórmula, generamos el correspondiente grafo usando `sy4ncl`. Luego, realizamos la búsqueda de automorfismos usando la herramienta `Bliss` [Junttila and Kaski, 2007]. `Bliss` toma la especificación de un grafo y retorna un conjunto de generadores para el grupo de automorfismos del grafo. Si el grafo tiene automorfismos no triviales, entonces reconstruimos las simetrías de la fórmula utilizando el mapeo entre nodos y literales generado por `sy4ncl`.

De ahora en mas llamamos *algoritmo de detección global (en capas)* o simplemente *detección global (en capas)* a la combinación del algoritmo global (en capas) de generación del grafo y la herramienta de detección de automorfismos.

Ejemplo 3.3. *Considere la fórmula* $\varphi = (p \vee \neg\Box p) \wedge (q \vee \neg\Box q)$. *La Figura 3.3a) muestra su representación en formato* intohylo. *La Figura 3.3b) muestra el contenido del archivo de mapeo generado por* `sy4ncl`. *Este archivo mapea los literales en la fórmula con los nodos en el grafo. La Figura 3.3c) muestra el grafo generado en formato* `Bliss`. *La Figura 3.3d) muestra las estadísticas sobre el grafo. La Figura 3.3e) muestra la salida generada por* `Bliss` *para el grafo correspondiente a* φ. *La misma, muestra que* `Bliss` *ha encontrado un generador. Usando este generador y el archivo de mapeo podemos reconstruir la simetría* $(p\ q)(\neg p\ \neg q)$ *de la fórmula.*

3.4.2 *Conjuntos de Fórmulas*

Para la pruebas utilizamos dos conjuntos de fórmulas, uno estructurado y uno aleatorio. El conjunto de fórmulas estructurado contiene 4492 fórmulas: 378 fórmulas provenientes del Logics Workbench Benchmark para K (LWB_K) [Balsiger *et al.*, 2000] (distribuidas en 9 clases) y 4113 fórmulas provenientes de la QBF-LIB [Giunchiglia *et al.*, 2001] (distribuidas en 23 clases). Los problemas provenientes de la QBF-LIB, fueron traducidos a la lógica modal básica usando la herramienta `qbf2ml`[2]. Usamos dos traducciones de Quantified Boolean Formulas (QBF) a lógica modal básica: *Collapse*1 y *Collapse*2. Estas traducciones son variantes de la traducción de Ladner [Ladner, 1977], que reducen la profundidad modal de las fórmulas resultantes, generando fórmulas mas pequeñas. La diferencia entre las traducciones Collapse1 y Collapse2 reside en que la primera utiliza variables proposicionales auxiliares mientras que la segunda no lo hace. Referimos al lector a [Orbe, 2014] para una descripción más detallada de ambas traducciones.

El conjunto de fórmulas aleatorio contiene 19000 fórmulas generadas usando la herramienta hGen [Areces and Heguiabehere, 2003]. Para generar las fórmulas, fijamos una profundidad modal máxima (D) y el tamaño de las cláusulas en 3. Luego, las instancias son distribuidas en 10 conjuntos. Para cada conjunto fijamos

1 Descargar desde: `http://cs.famaf.unc.edu.ar/~ezequiel/resource/sy4ncl`
2 Descargar desde:`http://cs.famaf.unc.edu.ar/~ezequiel/resource/qbf2ml`

```
begin
P1 v -[R1]P1;
P2 v -[R1]P2
end
```

a) Formato `intohylo`

```
<md> <node_id> <lit>
0 2 6
0 1 2
0 -1 3
0 -2 7
```

b) Archivo de Mapeo

```
Generator: (1,5)(2,6)(3,7)(4,8)
Nodes:          3
Leaf nodes:     3
Bad nodes:      0
Canrep updates: 1
Generators:     1
Max level:      1
|Aut|:          2
Total time: 0.00 seconds
```

e) Salida de `Bliss`

```
p edge 8 8
n 1 1
n 2 4
n 3 4
n 4 3
n 5 1
n 6 4
n 7 4
n 8 3
e 2 3
e 1 2
e 1 4
e 4 2
e 6 7
e 5 6
e 5 8
e 8 6
```

c) Grafo en formato Bliss

```
Computation time: 0.00000 sec
Color count:[2,0,2,4]
|Nodes|: 8
|Edges|: 8
```

d) Estadísticas del grafo

Figura 3.3: Salida de la herramienta `sy4ncl`.

el número de variables proposicionales (N) (entre 10 y 500) y variamos el número de cláusulas (L) para obtener diferentes valores para el ratio cláusulas-a-variables (L/N). Este ratio es un buen indicador del estatus de satisfacibilidad de la fórmula: fórmulas con un valor pequeño de L/N usualmente son satisfacibles, mientras que fórmulas con un valor grande de L/N usualmente son no satisfacibles. Cada conjunto contiene 100 fórmulas para 19 valores diferentes del ratio L/N (desde 0,2 a 35).

Todos las pruebas fueron ejecutadas en un Intel Core i7 2,93GHz con 16GB de RAM y un timeout de 120 segundos para la creación del grafo y la detección de simetrías.

3.4.3 *Resultados*

La Tabla 3.1 presenta los resultados para el conjunto de fórmulas LWB_K usando ambos algoritmos de detección (global y en capas). Las columnas *#In*, *#To* y *#Sy* representan el número de instancias en el conjunto, el número de instancias que dieron timeout y el número de instancias con al menos una simetría (no trivial), res-

pectivamente. Las columnas T_G y T_S representan el tiempo, en segundos, para crear el grafo, y para buscar los automorfismos en todas las fórmulas, respectivamente.

	$\#In$	$\#To$	$\#Sy$	T_G	T_S
Global	378	0	135	9,83	1,18
En capas	378	0	208	9,80	1,80

Tabla 3.1: Simetrías en el LWB_K.

La tabla muestra la existencia de muchas instancias simétricas en el LWB_K y que el tiempo requerido para computar las simetrías (tiempo de construcción más tiempo de detección) es negligible. También confirma nuestra hipótesis de que la detección de simetrías en capas detecta más simetrías que la detección global. De hecho, usando detección en capas, podemos detectar 73 instancias simétricas más que usando detección global.

La Tabla 3.2 muestra resultados detallados para el LWB_K. La columna M es la mediana del número de generadores detectados en cada clase de problema. Este valor da una idea aproximada de cuán simétricas son las instancias en una determinada clase de problemas. Las clases de problemas terminadas en "`_n`" contienen instancias satisfacibles, mientras que aquellas terminadas en "`_p`" contienen instancias no satisfacibles.

La tabla muestra que el LWB_K presenta un comportamiento que se ajusta a nuestras expectativas: la existencia de simetrías esta determinada por la codificación de cada clase de problemas. Muchas clases (`k_branch`, `k_path`, `k_grz`, `k_ph` y `k_poly`) presentan un gran número de instancias simétricas, mientras que otras no exhiben ninguna (`k_d4`, `k_dum`, `k_t4p`) o muy pocas (`k_lin`). También debe notarse el efecto de usar detección en capas. En algunas clases (`k_branch`, `k_ph` y `k_poly`) ambos algoritmos de detección detectan prácticamente el mismo número de instancias simétricas, con el algoritmo de detección en capas detectando algunas simetrías mas por instancia que el algoritmo global. Sin embargo, en las clases `k_path` y `k_grz`, las diferencias son más evidentes, ya que con el algoritmo de detección en capas encontramos más simetrías que con el algoritmo global.

Ahora nos concentramos en las fórmulas provenientes de la QBF-LIB. La Tabla 3.3 resume los resultados obtenidos en la QBF-LIB usando los dos algoritmos para ambas traducciones.

La Tabla 3.3 muestra que el la QBF-LIB es altamente simétrica: usando la traducción Collapse2 con detección global obtenemos un 65 % de instancias con una o más simetrías y 18 de 23 clases con al menos una instancia simétrica. Estos números suben hasta un 94 % de instancias simétricas y 23 de 23 clases teniendo al menos una instancia simétrica cuando usamos la traducción Collapse1 con detección en capas.

Esta tabla también muestra cuan sensible es la detección de simetrías a la codificación de las fórmulas. Usando la traducción Collapse1 encontramos más simetrías usando la detección en capas que usando la detección global.

Por otra parte, en fórmulas traducidas usando la traducción Collapse2 ambos algoritmos detectan prácticamente el mismo número de instancias simétricas. Esto se puede explicar por el hecho de que la traducción Collapse1 usa variables auxiliares

		Global		En capas	
Clase	*#In*	*#Sy*	*M*	*#Sy*	*M*
k_branch_n	21	21	12	21	12
k_branch_p	21	21	11	21	11
k_d4_n	21	0	-	0	-
k_d4_p	21	0	-	0	-
k_dum_n	21	0	-	0	-
k_dum_p	21	0	-	0	-
k_grz_n	21	1	1	21	5
k_grz_p	21	1	1	21	3
k_lin_n	21	0	-	0	-
k_lin_p	21	1	1	1	1
k_path_n	21	4	1,5	21	36
k_path_p	21	5	2	21	33
k_ph_n	21	18	1	18	1
k_ph_p	21	21	1	21	1
k_poly_n	21	21	16	21	19
k_poly_p	21	21	16	21	17
k_t4p_n	21	0	-	0	-
k_t4p_p	21	0	-	0	-

Tabla 3.2: Simetrías en el LWB_K detalladas por clase de problema.

(para marcar los niveles en el modelo arbolar resultante), mientras que la traducción Collapse2 no lo hace. Como consecuencia, en las fórmulas obtenidas con la traducción Collapse1 existen más variables proposicionales que pueden ser permutadas. La tabla también muestra que, en términos de eficiencia, detectar simetrías en capas es más difícil que detectar simetrías globales en fórmulas traducidas con Collapse1, en particular en lo que respecta a la búsqueda de automorfismos.

Las Figuras 3.4 y 3.5 muestran gráficos de dispersión del tiempo de construcción del grafo y del tiempo de búsqueda de automorfismos, para ambos algoritmos, en instancias traducidas usando Collapse1. El eje x da el tiempo de detección para el caso global, mientras que el eje y da el tiempo de detección para el caso en capas. Cada punto representa una instancia y sus coordenadas horizontales y verticales representan el tiempo necesario para construir el grafo (o buscar automorfismos) en segundos. Puntos debajo de la diagonal representan instancias donde la detección en capas requiere menos tiempo que la detección global, mientras que los puntos por encima de la diagonal representan instancias donde la detección global requiere menos tiempo que la detección en capas. Los puntos en los extremos (superior y derecho) representan timeouts. Notar que se utiliza una escala logarítmica, por lo que una ganancia o degradación más hacia la derecha es exponencialmente más significativa.

		Collapse1				Collapse2			
	#In	*#To*	*#Sy*	T_G	T_S	*#To*	*#Sy*	T_G	T_S
Global	4113	22	2678	7330,55	57695,41	24	2676	6310,18	44022,18
En capas	4113	20	3874	7596,48	65012,79	20	2680	6342,06	39370,36

Tabla 3.3: Simetrías en la QBF-LIB.

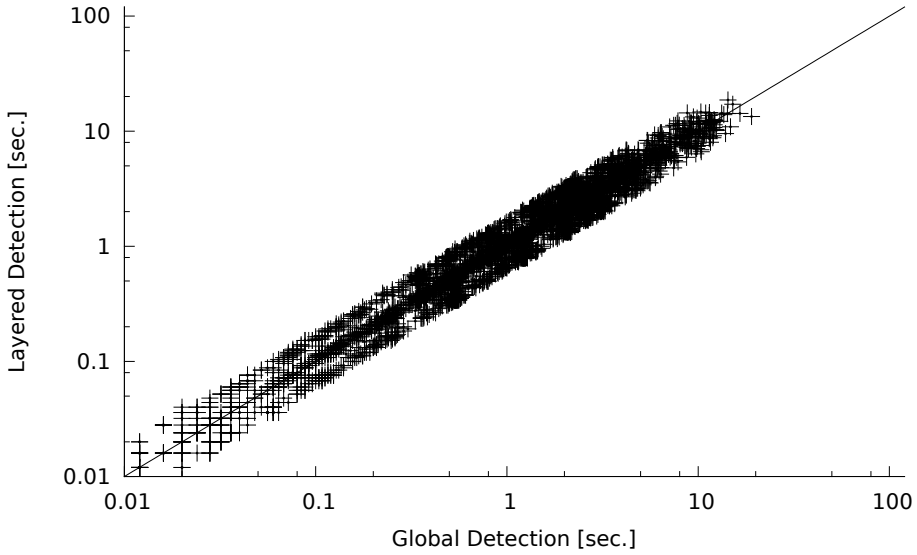


Figura 3.4: Tiempo de construcción del grafo en fórmulas Collapse1.

La Figura 3.4 muestra que en general no existen diferencias importantes en los tiempos requeridos para la construcción de los grafos para ambos algoritmos en fórmulas traducidas usando Collapse1. La Figura 3.5, por otra parte, muestra que la búsqueda de automorfismos es mas difícil en el caso de la detección en capas que en el caso de la detección global. Esto está en línea con el hecho de que la detección en capas implica grafos más grandes lo cual afecta directamente la performance de la herramientas de detección de grafos. Sin embargo, la Tabla 3.3 muestra un comportamiento diferente para fórmulas traducidas usando Collapse2. De hecho, en este caso, obtenemos mejor performance para la detección en capas que para la detección global. Las Figuras 3.6 y 3.7 muestran gráficos de dispersión para los tiempos de construcción y de detección para ambos algoritmos usando la traducción Collapse2.

La Figura 3.6 muestra que, de forma similar a lo que pasa con fórmulas Collapse1, no existen diferencias importantes en los tiempos de construcción de ambos algoritmos. Sin embargo, en la Figura 3.7, observamos que para un gran número de instancias, la detección de simetrías en capas es menos costosa que la detección de simetrías globales.

A primera vista, esto parece contradictorio ya que los grafos generados por el algoritmo de detección en capas son más grandes que los grafos generados por el algoritmo global, y es de esperarse un comportamiento similar al que obtuvimos para las instancias Collapse1. Sin embargo, este comportamiento puede ser explicado por el hecho de que, para las fórmulas Collapse2, los grafos generados por

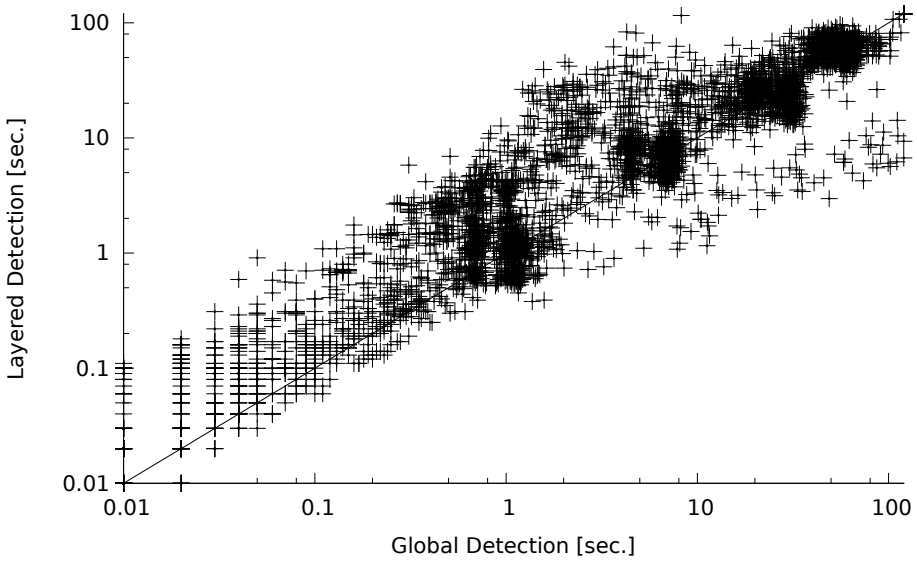


Figura 3.5: Tiempo de búsqueda de automorfismos en fórmulas Collapse1.

el algoritmo en capa son más dispersos (*sparser*) que los grafos generados por el algoritmo global. Un grafo es *disperso* si el grado promedio de sus nodos es mucho m'as chico que el número de nodos. En nuestro caso, para una fórmula Collapse2, su grafo en capas contiene más nodos que su grafo global, pero aproximadamente el mismo número de arcos. Esto reduce el grado promedio de los nodos, haciendo al grafo en capas más disperso que el grafo global, y es bien conocido que la performance de las herramientas de detección de grafos está directamente afectada por la dispersión de los grafos de entrada [Junttila and Kaski, 2007; Darga *et al.*, 2004; Darga *et al.*, 2008; Katebi *et al.*, 2012].

Finalmente probamos los algoritmos de detección en conjuntos de fórmulas aleatorias para observar con que frecuencia obtenemos simetrías en ellas. Las Figuras 3.8 y 3.9 muestran el porcentaje de instancias simétricas para cada valor del ratio L/N usando detección global y en capas respectivamente.

Estos gráficos muestran que para valores pequeños de L/N es fácil encontrar instancias simétrica aún en instancias aleatorias. A medida que incrementamos el valor del ratio, el número de instancias simétricas decrece rápidamente. De nuevo, esto coincide con nuestras expectativas: grandes valores de L/N resultan de la presencia de un gran número de cláusulas en las fórmulas, reduciendo la posibilidad de simetrías.

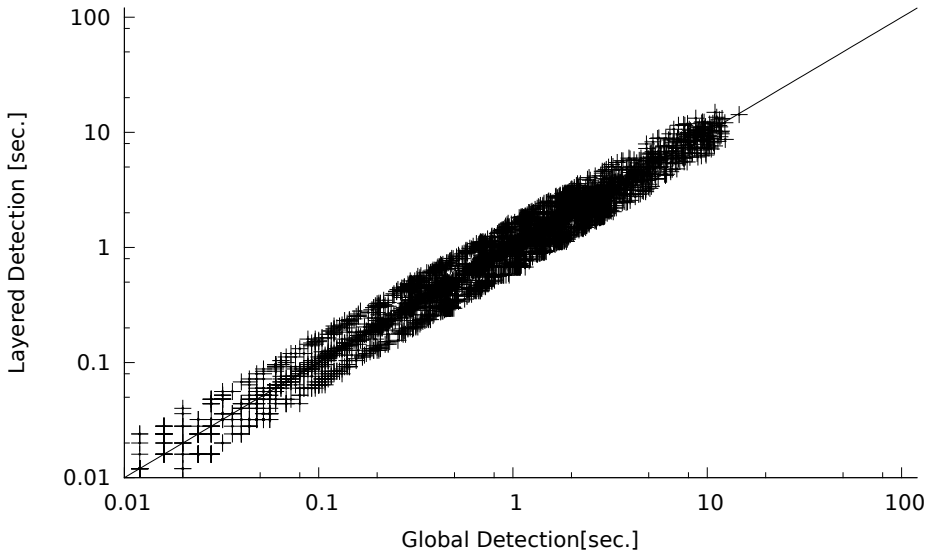


Figura 3.6: Tiempo de construcción del grafo en fórmulas Collapse2.

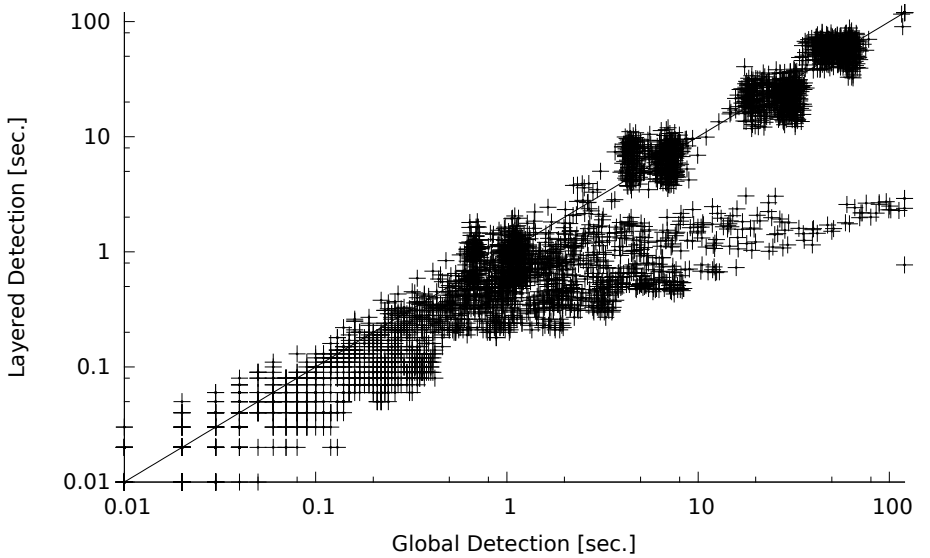


Figura 3.7: Tiempo de búsqueda de automorfismos en fórmulas Collapse2.

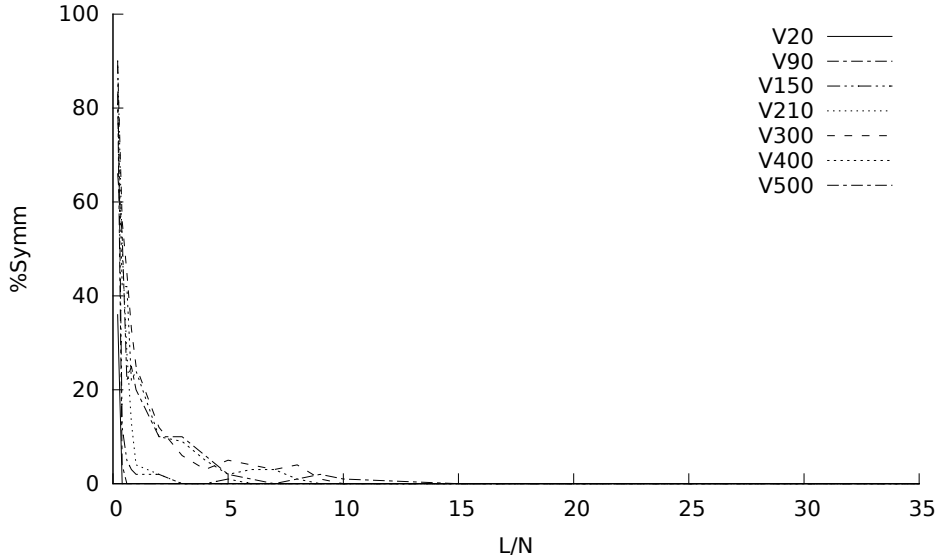


Figura 3.8: Porcentaje de fórmulas aleatorias simétricas usando detección global.

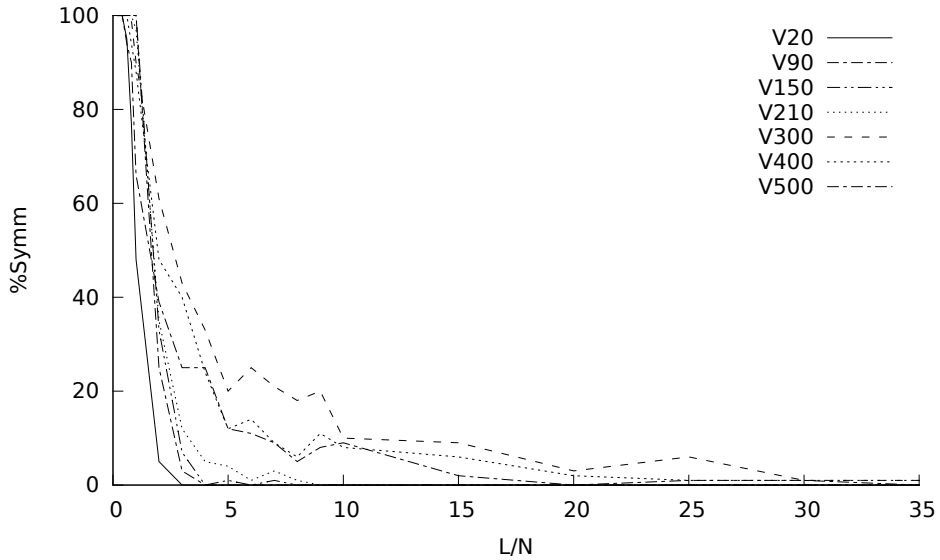


Figura 3.9: Porcentaje de fórmulas aleatorias simétricas usando detección en capas.

4

DETECCIÓN DE SIMETRÍAS PARA SATISFACIBILIDAD MODULO TEORÍAS

En el capítulo previo presentamos una técnica basada en grafos para la detección de simetrías en fórmulas modales. En este capítulo extendemos esta técnica a fórmulas SMT.

Comenzamos por dar un breve repaso sobre simetrías en SMT (Sección 4.1). Luego introducimos las definiciones y notación necesarias para el resto del capítulo (Sección 4.2). A continuación presentamos el algoritmo para la detección de simetrías en fórmulas SMT y probamos su corrección (Sección 4.3). Finalmente evaluamos empíricamente el algoritmo en varios conjuntos de pruebas provenientes de la SMT-LIB (Sección 4.4).

En lo que sigue, asumimos conocimientos básicos sobre SMT y referimos al lector a [Sebastiani, 2007; Barrett *et al.*, 2009] por más detalles sobre el tema.

Los resultados presentados en este capítulo fueron publicados en [Areces *et al.*, 2013].

4.1 SIMETRÍAS EN SMT

Hasta donde conocemos, la investigación sobre simetrías en SMT es reciente y las mismas no son completamente explotadas en los solvers de SMT.

El primer intento por utilizar simetrías en SMT es el de [Audemard *et al.*, 2002], en el cual las mismas son utilizadas como una técnica de simplificación en el chequeo de modelos basado en SMT (SMT-based model checking).

En [Roe, 2006], un solver SMT introduce predicados de ruptura de simetrías en la fórmula de entrada en una etapa de preprocesamiento. Para ello, todos los pares de variables simétricas son detectados. Dada una fórmula φ, un par de variables (a, b) que ocurre en φ es un par de variables simétricas si cuando reemplazamos todas las instancias en φ de a por b y las de b por a, obtenemos nuevamente φ. Una vez que un par simétrico es computado, pares simétricos de átomos son identificados. Dos átomos p_1 y p_2 son simétricos si existe un conjunto de pares de variables simétricos $\{(a_1, b_1), \ldots, (a_n, b_n)\}$ tal que p_1 puede ser transformado en p_2 simplemente reemplazando cada a_i por b_i y viceversa. Luego un grupo de predicados simétricos es computado. Un grupo de predicados simétricos es un conjunto de dos o más átomos tales que cualesquiera dos átomos en el grupo son simétricos. Finalmente, predicados de ruptura de simetrías, similares a los introducidos en [Aloul *et al.*, 2003b], son agregados a la fórmula original. De acuerdo a lo que se reporta en [Roe, 2006], esta técnica sólo funciona para fórmulas en la clase QF_UF, es decir, en fórmulas sin cuantificadores construidas sobre una signatura con símbolos y tipos no interpretados), sin embargo, no se proveen resultados experimentales para evaluar su efectividad.

Recientemente, en [Déharbe *et al.*, 2011], un algoritmo para detectar y romper simetrías es presentado e implementado en el solver veriT [Bouton *et al.*, 2009].

El algoritmo trabaja detectando grupos completos de simetrías de constantes no interpretadas y agregando predicados de ruptura de simetrías.

Este trabajo se basa en la observación de que, en SMT, una fuente frecuente de simetrías se da cuando un término toma sus valores de un conjunto finito de elementos completamente simétricos. De esta forma, dada una fórmula φ, simétrica (o invariante) con respecto a todas las permutaciones de un conjunto de constantes no interpretadas $c_0, \ldots, c_n$, para cualquier modelo $\mathcal{M}$ de φ, si el término t no contiene estas constantes y $\mathcal{M}$ satisface $t = c_i$ para algún $i \in \{1, \ldots, n\}$, luego debe existir un modelo en el cual t sea igual a c_0. De esta forma, mientras verificamos si una fórmula es no satisfacible alcanzaría con verificar solamente aquellos modelos donde t y c_0 tengan el mismo valor.

La detección de simetrías se realiza primero "adivinando" una permutación y luego chequeando que la misma es una simetría de la fórmula. Resultados experimentales confirman el éxito de ésta técnica, en particular en problemas de la clase QF_UF. Sin embargo, esta técnica es limitada en el sentido de que solo funciona con grupos de simetrías completos, y que no toma en cuenta aquellas simetrías que involucran otros tipos de símbolos, como símbolos de predicados, funciones, símbolos interpretados o cuantificadores.

4.2 DEFINICIONES

Ahora definiremos el lenguaje que utilizaremos en el resto del capítulo. Es común introducir SMT usando un lenguaje de primer orden estándar. Sin embargo, los problemas en SMT (en particular aquellos en la SMT-LIB [Barrett *et al.*, 2010b]) se expresan usualmente en un lenguaje de primer orden tipado (many-sorted). En estos lenguajes, cada término está *tipado*, y cada tipo es denotado con un símbolo de tipo.

Definición 4.1 (Signatura Tipada)**.** *Una* signatura tipada *(*many-sorted signature*) es una tupla* $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}, ar, r \rangle$ *donde*

- $\mathcal{S}$ *es un conjunto contable no vacío de tipos disjuntos que contiene el tipo* `Bool`*.*
- $\mathcal{V}$ *es la unión (contable) de conjuntos disjuntos* $\mathcal{V}_\tau$ *de variables de tipo* $\tau \in \mathcal{S}$*.*
- $\mathcal{F}$ *es un conjunto contable infinito de símbolos de función que contiene los símbolos* $=, \wedge, \neg, \forall_\tau$*, y* $\exists_\tau$ *para todo* $\tau \in \mathcal{S}$*.*
- $ar : \mathcal{F} \mapsto \mathbb{N}$ *es una función total llamada* aridad*, y* $r : \mathcal{F} \mapsto S^n \times S$ *es una función llamada* rank *tales que:*
  - $r(\wedge) = (\texttt{Bool}, \texttt{Bool}, \texttt{Bool})$ *y* $ar(\wedge) = 2$*.*
  - $r(\neg) = (\texttt{Bool}, \texttt{Bool})$ *y* $ar(\neg) = 1$*.*
  - $r(=) = (\tau, \tau, \texttt{Bool})$ *y* $ar(=) = 2$ *para todo* $\tau \in \mathcal{S}$*.*
  - $r(\forall_\tau) = (\tau, \texttt{Bool}, \texttt{Bool})$ *y* $ar(\forall_\tau) = 2$ *para todo* $\tau \in \mathcal{S}$*.*
  - $r(\exists_\tau) = (\tau, \texttt{Bool}, \texttt{Bool})$ *y* $ar(\exists_\tau) = 2$ *para todo* $\tau \in \mathcal{S}$*.*
  - $r(f) \in S^{ar(f)} \times S$ *para todo* $f \in \mathcal{F} \backslash \{=, \wedge, \neg, \forall_\tau, \exists_\tau\}$*.*

Notar que el *rank* de un símbolo de función específica, en orden, los tipos esperados para los símbolos de los argumentos y el resultado.

Como es usual, llamamos *constantes* a los símbolos de función con aridad 0.

En nuestro lenguaje los términos se construyen con variables de $\mathcal{V}$ y símbolos de función de $\mathcal{F}$.

Definición 4.2 (Σ-términos). *Sea $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}, ar, r \rangle$ una signatura. El conjunto de Σ-términos bien tipados, es el conjunto más chico definido por:*

$$\begin{array}{rcll} T_\Sigma & ::= & x & \text{donde } x \in \mathcal{V} \\ & | & f(t_1, \ldots, t_n) & \text{donde } r(f) = (\tau_1, \ldots, \tau_n, \tau) \\ & & & \text{y por cada } i \in \{1, \ldots, n\}, t_i \text{ tiene tipo } \tau_i. \end{array}$$

Consideramos una variable como libre o acotada de la forma usual. Un término es cerrado *si no contiene variables libres, y es* abierto *en caso contrario. Un término es* base (ground) *si no contiene variables, ya sean libres o acotadas.*

Definición 4.3 (Σ-fórmulas). *Sea $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}, ar, r \rangle$ una signatura. El conjunto de las Σ-fórmulas se define como el conjunto de términos bien tipados con tipo* `Bool`.

Notar que por simplicidad, el lenguaje definido no contiene símbolos lógicos. Los conectivos lógicos para la negación, conjunción, el símbolo de igualdad y los cuantificadores, son considerados como símbolos de función con un rank y tipo predefinidos.

Asignamos un significado a las Σ-fórmulas mediante Σ-estructuras.

Definición 4.4 (Σ-estructura). *Una estructura $\mathcal{I}$ para una signatura $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}, ar, r \rangle$ (o Σ-estructura), es un par $\mathcal{I} = \langle D, (\_)^{\mathcal{I}} \rangle$ donde D asigna un dominio no vacío D_τ a cada tipo $\tau \in \mathcal{S}$ y $(\_)^{\mathcal{I}}$ asigna un significado a cada variable y símbolo de función.*

Por extensión una Σ-estructura define un valor $\mathcal{I}[t]$ en D_τ para cada término de tipo τ, y un valor de verdad $\mathcal{I}[\varphi]$ en $\{true, false\}$ para cada fórmula φ.

Decimos que una Σ-fórmula cerrada φ es *satisfacible* si existe una Σ-estructura $\mathcal{I}$ que haga verdadera a la fórmula (notación $\mathcal{I} \models \varphi$). Decimos que φ es *válida* si para todas las Σ-estructuras $\mathcal{I}$, $\mathcal{I} \models \varphi$. Una Σ-estructura $\mathcal{I}$ satisface un conjunto S de Σ-fórmulas ($\mathcal{I} \models S$) si $\mathcal{I} \models \varphi$ para toda $\varphi \in S$.

Al igual que en lenguajes más convencionales, este lenguaje es en realidad una familia de lenguajes parametrizados por una signatura Σ. Sin embargo, cuando trabajamos en el contexto de una teoría $\mathcal{T}$, la signatura específica es definida conjuntamente por la declaración de $\mathcal{T}$ más todo tipo y símbolo de función adicional contenido en la fórmula. Un símbolo de función es *interpretado* si es definido por $\mathcal{T}$, o *no interpretado* en caso contrario.

Notar que, como se sigue de nuestra definición de Σ-términos, los términos bien tipados tienen un único tipo. Por lo tanto podemos definir una función de *tipo* que nos devuelva el tipo de cada término y símbolo de función.

Definición 4.5 (Función de Tipo). *Sea $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}, ar, r \rangle$ una signatura. Definimos una* función de tipo *$sort : \mathcal{V} \cup \mathcal{F} \cup T_\Sigma \mapsto \mathcal{S}$ que mapea cada variable, símbolo de función y Σ-término a su correspondiente tipo.*

En lo que sigue, por simplicidad, omitiremos el prefijo Σ para referirnos a los términos, estructuras, fórmulas, etc. Además, nos referiremos a toda fórmula bien formada en nuestro lenguaje de primer orden tipado como una *fórmula SMT*.

4.3 DETECTANDO SIMETRÍAS

Ahora presentamos el algoritmo de detección de simetrías para fórmulas SMT. Este algoritmo, dada una fórmula, construye el grafo sintáctico dirigido, acíclico y coloreado más algunos nodos adicionales. El coloreo del grafo es definido mediante una *función de tipado*.

Definición 4.6 (Función de Tipado). *Sea $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}, ar, r \rangle$ una signatura y $\mathcal{T}$ una teoría de primer orden sobre Σ. Con $\mathcal{F}_{\mathcal{T}}$ y F_U denotamos los conjuntos de todos los símbolos* interpretados *y* no interpretados *respectivamente. Sea D un conjunto contable infinito y, sea e un elemento de D. Definimos las funciones inyectivas $a : \mathcal{S} \mapsto A$, $b : \mathcal{S}^n \times \mathcal{S} \mapsto B$, y $c : \mathcal{F}_{\mathcal{T}} \mapsto C$ tales que A, B, y C son una partición de $D \backslash \{e\}$ (es decir, A, B y C son no vacías, $A \cap B \cap C = \emptyset$ y $A \cup B \cup C = D \backslash \{e\}$). Definimos la* función de tipado *$t : \mathcal{F} \cup T_\Sigma \cup \{\star\} \mapsto D$ como:*

$$t(s) = \begin{cases} e & \text{si } s = \star. \\ a(sort(s)) & \text{si } s \text{ es un término o un símbolo de constante no interpretado.} \\ b(r(s)) & \text{si } s \text{ es un símbolo de función no interpretado con } ar(s) > 0. \\ c(s) & \text{si } s \text{ es un símbolo de función interpretado.} \end{cases}$$

Notar como la función de tipado maneja los diferentes elementos de una fórmula. Asigna un color a los términos y símbolos de constantes no interpretados basado en el tipo de los mismos, a los símbolos de función no interpretados basado en su rank, y a los símbolos interpretados un único color independientemente de su tipo y rank. En lo que sigue, asumimos que los colores están representados por números naturales, es decir $D = \mathbb{N}$ y $e = 0$.

Definición 4.7 (Algoritmo de Detección). *Sea φ una fórmula SMT, y t una función de tipado. Sea $\mathcal{F}(\varphi)$ el conjunto de símbolos de función que ocurren en φ. El grafo dirigido coloreado $G(\varphi) = (V, E)$ se construye recursivamente de la siguiente forma:*

i) Por cada símbolo $s \in \mathcal{F}(\varphi)$:
   a) *Agregar un nodo* símbolo *de color $t(s)$.*

ii) Por cada término $f(t_1, \ldots, t_n)$, en φ, de aridad $n > 0$:
   a) *Agregar un nodo de* ocurrencia *de color $t(f(t_1, \ldots, t_n))$.*
   b) *Agregar un arco desde el nodo de ocurrencia al nodo símbolo de f.*
   c) *Si la función es conmutativa (e.j., $\wedge$, $\vee$, $\leftrightarrow$, $=$, $+$, $*$):*
      1) *Agregar un arco desde el nodo de ocurrencia al nodo raíz del grafo $G(t_i)$ para $1 \leq i \leq n$.*
   d) *Si la función es no conmutativa:*
      1) *Por cada argumento t_i, agregar un nodo* argumento *de color $t(\star)$ y un arco desde este nodo al nodo raíz de $G(t_i)$.*
      2) *Agregar un arco desde el nodo argumento de t_i al nodo argumento de t_{i+1} ($1 \leq i < n$). Estos arcos representan el orden de los argumentos en $f(t_1, \ldots, t_n)$.*

3) *Agregar un arco desde el nodo de ocurrencia al nodo argumento de* t_1.

iii) *Por cada término* $Qx.t$ $Q \in \{\forall, \exists\}$:

a) *Proceder como si fuera un símbolo de función conmutativo, e. j., la función* $Q(x,t)$.

Ejemplo 4.1. *Considere la fórmula* $\varphi = f(a,b) \vee f(b,a)$ *donde* f, a, *y* b *son símbolos no interpretados tales que* $sort(a) = sort(b) = U$ *y* $r(f) = (U, U, \texttt{Bool})$. *La Figura 4.1 muestra el grafo coloreado asociado* $G(\varphi)$, *construido asumiendo que* f *es conmutativa (los colores se presentan mediante las distintas formas de los nodos en la figura).*

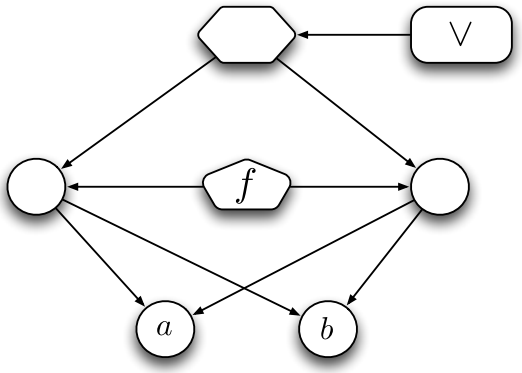


Figura 4.1: Grafo de $f(a,b) \vee f(b,a)$ (f conmutativa).

Ejemplo 4.2. *Considere la fórmula del Ejemplo 4.1, pero ahora asumamos que* f *es no conmutativa. La Figura 4.2 muestra el grafo coloreado asociado* $G(\varphi)$. *Notar como los nodos argumentos (nodos triangulares en la figura) son utilizados. Para el término* $f(a,b)$ *el nodo argumento de* a, *el primer argumento, es directamente conectado al nodo de ocurrencia y al nodo argumento correspondiente a* b. *Para el término* $f(b,a)$ *esto se invierte. El nodo argumento de* b *está directamente conectado al nodo de ocurrencia y al nodo argumento correspondiente a* a.

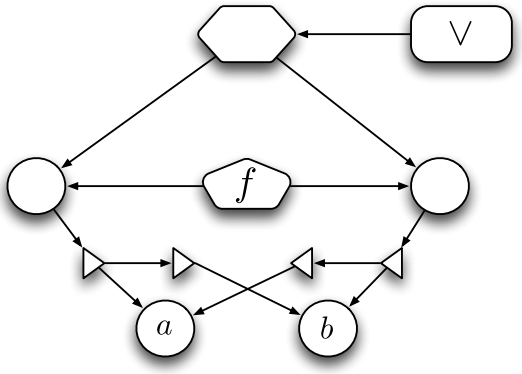


Figura 4.2: Grafo de $f(a,b) \vee f(b,a)$ (f es no conmutativa).

Ahora probamos que el algoritmo es correcto.

Teorema 4.1. *Sea φ una fórmula SMT y $G(\varphi) = (V, E)$ el grafo coloreado construido a partir de la Definición 4.7. Luego, todo automorfismo de $G(\varphi)$ es una simetría de la fórmula φ.*

Demostración. La prueba se sigue por inducción estructural y las siguientes observaciones.

- El grafo $G(\varphi)$ es al grafo sintáctico acíclico y dirigido de φ más nodos adicionales.
- Para términos, $f(t_1, \ldots, t_n)$ de aridad > 0, el coloreo de los nodos, y la combinación de los nodos raíz y los nodos símbolos, aseguran que solo nodos símbolos del mismo tipo, es decir, de la misma aridad y con los mismos tipos de argumentos, y con el mismo número de ocurrencias pueden ser permutados.
- Para términos sin argumentos (constantes y predicados), el coloreo de los nodos símbolos y la existencia de nodos argumentos aseguran que solo símbolos del mismo tipo y que ocurran el mismo número de veces en la misma posición como argumentos pueden ser permutados.

Finalmente, para reconstruir la simetría de la fórmula a partir de un automorfismo del grafo, solo tenemos que restringir el automorfismo a los nodos símbolos. □

Notar que la vuelta del Teorema 4.1 es falsa: la construcción propuesta no permite detectar todas las simetrías de la fórmula. Por ejemplo, considere la fórmula $\varphi = f(a, b) \wedge g(c, d)$ donde a, d son del mismo tipo y b, c de otro tipo (con tipos apropiados para f y g). La permutación $\sigma = (f\ g)(a\ c)(b\ d)$ es una simetría de φ. Sin embargo, no la podemos detectar en el grafo $G(\varphi)$. Esto se debe al hecho de que los nodos símbolo son coloreados en función al tipo del símbolo, lo cual evita que se detecten automorfismos que permuten símbolos de distinto tipo. Sin embargo, desde un punto de vista práctico, las simetrías que permutan símbolos de diferentes tipos son poco naturales y no ocurren frecuentemente.

4.4 EVALUACIÓN EXPERIMENTAL

Ahora presentamos datos experimentales sobre la existencia de simetrías en fórmulas SMT y cuan difícil es detectarlas.

4.4.1 *Implementación*

Para realizar la evaluación implementamos el algoritmo de la Definición 4.7 en la herramienta SyMT[1].

SyMT es una herramienta de línea de comando implementada en C. La misma toma en cuenta la conmutatividad de la conjunción, disyunción, adición, multiplicación e igualdad. Dada una fórmula SMT, SyMT crea un grafo coloreado y usando

1 Descargar desde: http://www.verit-solver.org/SyMT/

un componente de detección de automorfismos, detecta el grupo de automorfismos del grafo coloreado. En particular, SyMT usa `Saucy` 3.0 [Katebi *et al.*, 2012] como el componente de detección de automorfismos. La integración con `Saucy` es realizada a través de su API de C

SyMT también provee servicios de simplificación sobre la fórmula de entrada, algunos de los cuales involucran razonamiento en la teoría (lo cual puede fallar en instancias grandes). Simplificar la fórmula de entrada es importante ya que se pueden descubrir simetrías escondidas y a la vez remover simetrías triviales (es decir, simetrías que no involucran símbolos no interpretados). Los servicios de simplificación incluyen reescritura simple, simplificación de literales deducidos, y normalización de términos y fórmulas. En la herramienta estos servicios se activan con la opción `-s`.

Para mejorar la legibilidad del grupo de simetrías de la fórmula de entrada, se desarrollaron dos técnicas de post-procesamiento para identificar subgrupos que sean grupos de permutación completos sobre un subconjunto de los símbolos involucrados en la simetría. La primer técnica esta basada en el algoritmo Schreir-Sims [Seress, 2003; Rehn and Schürmann, 2010], un algoritmo polinomial que, entre otras cosas, hace posible verificar si una permutación dada pertenece al grupo generado por un conjunto de generadores. Usamos este algoritmo para verificar, para cada órbita, si el grupo completo de permutaciones de los elementos en la órbita es un subgrupo del grupo de simetrías. De hecho, para verificar si un grupo admite como subgrupo al grupo completo de permutaciones de estos elementos, alcanza con verificar la pertenencia de dos permutaciones, e. j., una permutación arbitraria entre dos elementos, y una permutación cíclica que involucre a todos los elementos en la órbita.

Aunque polinomial, la técnica descripta es usualmente extremadamente lenta. Para solucionar este problema, desarrollamos una segunda técnica, basada en heurísticas. La misma, en esencia, particiona los símbolos (usando una estructura de datos union-find) en clases tales que dos símbolos en la misma clase son simétricos. Los generadores son usados para unir las clases en la partición, y la partición es utilizada para simplificar los generadores. En la práctica, esta técnica funciona bien en la mayoría de los casos, aunque puede fallar en detectar subgrupos completos de permutaciones en algunos casos. Estas técnicas de post-procesamiento se pueden activar con las opciones `-R` y `-r` respectivamente.

Ejemplo 4.3. *La línea de comando y la salida de SyMT para una fórmula de la clase QF_UF es la siguiente:*

```
./SyMT -s -r smt-lib2/QF_UF/NEQ/NEQ004_size4.smt2
[c_0 c_1 c_2 c_3]
(p7 p9)(c12 c13)
```

SyMT encuentra los generadores para el grupo de simetrías. Luego detecta, usando las técnicas de post-procesamiento descriptas, que existe un subgrupo completo de permutaciones para las constantes `c_0`, `c_1`, `c_2`, `c_3`, *y otra simetría que permuta los símbolos de predicado binarios* `p7` *y* `p9` *junto con* `c12` *y* `c13`.

4.4.2 *Resultados*

Probamos SyMT en 19 clases de problemas[2] de la SMT-LIB [Barrett *et al.*, 2010a] para investigar la existencia de simetrías y evaluar la eficiencia de la herramienta. Todos los tests fueron ejecutados en un Intel Xeon X3440 con 16GB. Se probaron tres configuraciones diferentes de SyMT. La Configuración 1 no tiene simplificaciones: la fórmula es parseada y convertida en un grafo para la detección de los automorfismos. La Configuración 2 usa simplificaciones sintácticas triviales. La Configuración 3 habilita simplificaciones más potentes, usando el motor SMT, e. j., simplificación de átomos implicados por cláusulas unitarias. La Configuración 2 puede fallar (sin reportar simetrías) dado que la complejidad en tiempo del algoritmo de simplificación no es lineal con respecto al tamaño de la fórmula de entrada. Sin embargo, usualmente revela simetrías que están escondidas por el ruido sintáctico que se remueve fácilmente usando el procedimiento de simplificación. La Configuración 3 es propensa a fallar en fórmulas de gran tamaño, pero también puede revelar simetrías escondidas. El proceso de simplificación puede, a veces, reducir una fórmula a falso, en cuyo caso no se reportan simetrías. El timeout (relevante para las configuraciones 2 y 3) se establece en 30 segundos.

De las 19 clases analizadas, tres (LRA, QF_UFLRA, QF_UFNRA) no presentan simetrías con SyMT. De las cinco fórmulas en UFLRA, una tiene simetrías. Las otras 15 clases presentan un número significativo de simetrías en al menos una de las configuraciones testeadas. La Tabla 4.1 resume los resultados obtenidos para estas 15 clases. Para cada clase reportamos el número de instancias (#Inst), el número de instancias que tienen simetrías en las distintas configuraciones (#Sym[1], #Sym[2] y #Sym[3]), el número de instancias que tienen simetrías en al menos una de las configuraciones (#Sym[P]), el promedio del logaritmo en base 2 del tamaño del grupo de simetrías (Avg[GS]) para la Configuración 1, y el tiempo total, en segundos, requerido para analizar todas las instancias (Time) para la Configuración 1.

La Tabla 4.1 muestra claramente que la SMT-LIB tiene muchas fórmulas altamente simétricas en la mayor parte de las clases que la componen. Además, un estudio en profundidad de las simetrías detectadas, revela que éstas proveen información que dependen fuertemente de la clase de fórmulas en cuestión. Por ejemplo, muchas de las simetrías en fórmulas cuantificadas (e. j., AUFLIA) se deben a axiomatizaciones repetidas que no se usan. Mientras que las simetrías en los problemas `FISCHER` (QF_LIA, QF_IDL), los cuales han sido automáticamente generados a partir de algoritmos distribuidos de verificación de modelos acotados, revelan que algunos de los procesos son simétricos.

El tiempo acumulado para construir el grafo y detectar las simetrías es negligible en todas las clases. No presentamos los tiempos para las otras configuraciones dado que existen timeouts y el time se gasta mayormente en los módulos de simplificación, por lo cual estos números proveen poca información sobre el tiempo de detección de simetrías en si mismo.

2 Bit vectors no son soportados por nuestro parser.

Clase	#Inst	#Sym[1]	#Sym[2]	#Sym[3]	#Sym[P]	Avg[GS]	Time
AUFLIA	6480	6212	6231	5941	6258	134,00	378,79
AUFLIRA	19917	15779	16475	12500	16476	1,08	9,13
AUFNIRA	989	985	985	923	985	1,00	0,41
QF_AUFLIA	1140	2	71	77	78	1,00	0,72
QF_AX	551	22	22	22	22	1,00	0,37
QF_IDL	1749	348	526	683	756	12745,43	327,95
QF_LIA	5938	728	1172	524	1200	104,55	486,19
QF_LRA	634	73	150	208	210	110,49	29,06
QF_NIA	530	169	169	168	169	5,92	3,92
QF_NRA	166	9	43	43	43	1,00	0,23
QF_RDL	204	0	0	24	24	0,00	10,13
QF_UF	6639	250	3638	375	3638	44,00	34,58
QF_UFIDL	431	19	175	186	189	1,00	2,70
QF_UFLIA	564	0	198	198	198	0,00	0,45
UFNIA	1796	1062	1061	1058	1070	47,08	543,26

Tabla 4.1: Simetrías en la SMT-LIB.

5

SIMETRÍAS EN UN TABLEAUX MODAL

En este capítulo presentamos una técnica para utilizar información de simetrías en un tableaux modal. La técnica consiste en un mecanismo de bloqueo que aprovecha la información de simetrías de la fórmula de entrada para restringir la aplicación de la regla $(\Diamond)$.

Comenzamos por presentar las nociones básicas sobre tableaux etiquetados para la lógica modal básica (Sección 5.1). Luego presentamos los fundamentos teóricos del mecanismo de bloqueo y probamos su corrección (Sección 5.2). Finalmente presentamos resultados experimentales que muestran la aplicabilidad de este mecanismo (Sección 5.3).

Los resultados presentados en este capítulo fueron publicados en [Areces and Orbe, 2013].

5.1 TABLEAUX PARA LA LÓGICA MODAL BÁSICA

En esta sección presentamos un cálculo de tableaux etiquetado para la lógica modal básica [Blackburn *et al.*, 2006].

El tableaux semántico, o tableaux, es un procedimiento de prueba que decide satisfacibilidad de una fórmula basado en la satisfacibilidad de sus subfórmulas. Fue introducido en [Beth, 1959], y tomó su forma actual de forma independiente en [Lis, 1960] y [Smullyan, 1968]. Actualmente, es el procedimiento de prueba más popular para las lógicas modales debido a sus exitosas implementaciones y su flexibilidad para adaptarse a diferentes lógicas [D'Agostino, 1999].

Los sistemas de tableaux etiquetados para la lógica modal básica fueron introducidos en [Fitting, 1972; Fitting, 1983], pero tomaron su forma actual en [Massacci, 1994; Goré, 1999]. Los tableaux etiquetados están fuertemente relacionados con los tableaux proposicionales: son conjuntos de fórmulas que describen parcialmente un modelo. Sin embargo, en vez de tomar en cuenta los valores de verdad de los símbolos proposicionales en "un solo mundo", los consideran en un número arbitrario de mundos conectados, y cada fórmula es etiquetada con el mundo en el cual debe ser verdadera.

Definamos ahora formalmente el tableaux modal. Sea PREF un conjunto infinito, no-vacío de *prefijos*. Acá consideramos $\mathsf{PREF} = \mathbb{N}$. Asumiremos que todas las fórmulas están en CNF modal (Definición 2.2).

Definición 5.1 (Fórmulas prefijadas)**.** *Dada φ una fórmula en CNF modal, C una cláusula modal y $\alpha \in \mathsf{PREF}$ llamamos a $\alpha{:}\varphi$ y $\alpha{:}C$* fórmulas prefijadas. *La interpretación de una fórmula prefijada $\alpha{:}F$ es que F es verdadera en el estado denotado por α.*

Definición 5.2 (Declaraciones de Accesibilidad)**.** *Dados $\alpha, \alpha' \in \mathsf{PREF}$ llamamos a $\alpha R \alpha'$ una* declaración de accesibilidad. *La interpretación de $\alpha R \alpha'$ es que el estado denotado por α' es accesible desde α vía R.*

$$\frac{\alpha{:}\varphi}{\alpha{:}C_i,}\ (\wedge) \qquad \forall C_i \in \varphi \qquad \frac{\alpha{:}C}{\alpha{:}l_1 \mid \ldots \mid \alpha{:}l_n,}\ (\vee) \quad \forall l_i \in C$$

$$\frac{\alpha{:}\neg\Box C}{\alpha R\alpha',\ \alpha'{:}{\sim}C,}\ (\Diamond)^1 \qquad \frac{\alpha{:}\Box C, \alpha R\alpha'}{\alpha' : C}\ (\Box)$$

[1] $\sim C$ es la CNF de la negación de C. El prefijo α' es nuevo en el tableau.

Figura 5.1: Cálculo de tableaux etiquetado para la lógica modal básica.

Un tableaux para una fórmula en CNF modal φ es un árbol cuyos nodos están decorados con fórmulas prefijadas y declaraciones de accesibilidad, tal que el nodo raíz es $0{:}\varphi$. Además, requerimos que los nodos adicionales en el árbol sean creados de acuerdo con las reglas de la Figura 5.1, donde φ es una fórmula en CNF modal y C una cláusula modal.

Las reglas se interpretan de la siguiente forma: si los antecedentes de una regla aparecen en los nodos de una rama del tableau, la rama es extendida de acuerdo a las fórmulas en el consecuente de la regla. Para el caso de la regla $(\vee)$, n sucesores inmediatos del último nodo en la rama son creados. En todos los otros casos, solo un sucesor es creado, el cual es decorado con las fórmulas indicadas. Para asegurar terminación, requerimos que los nodos sean creados solamente si agregan al menos una fórmula o declaración de accesibilidad que no este presente en la rama. Además, la regla $(\Diamond)$ solo puede ser aplicada una vez a cada fórmula de la forma $\alpha{:}\neg\Box C$ en la rama.

Definición 5.3 (Rama Cerrada, Abierta y Saturada)**.** *Una rama está* cerrada *si contiene al mismo tiempo $\alpha{:}p$ y $\alpha{:}\neg p$, y esta* abierta *en caso contrario. Decimos que una rama está* saturada *si no se puede aplicar ninguna regla mas.*

Sea $\mathsf{Tab}(\varphi)$ el conjunto de tableaux para φ cuyas ramas están todas saturadas. El siguiente resultado clásico establece que el cálculo de tableaux definido es un procedimiento de decisión para satisfacibilidad de fórmulas en la lógica modal básica.

Teorema 5.1. *Para toda fórmula φ de la lógica modal básica, todo tableau $\mathsf{T} \in \mathsf{Tab}(\varphi)$ es finito. Además, T tiene una rama abierta y saturada si y solo si φ es satisfacible.*

5.2 BLOQUEO SIMÉTRICO

El cálculo de tableaux de la Figura 5.1 termina debido a que restringimos la aplicación de la regla $(\Diamond)$ de forma tal que solo puede ser aplicada una vez a cada fórmula de la forma $\alpha{:}\neg\Box C$.

Estamos interesados en restringir aún más la aplicación de la regla $(\Diamond)$ de forma tal de que solo sea aplicable a una fórmula $\alpha{:}\neg\Box C$ si no ha sido aplicada anteriormente a una fórmula simétrica. Llamamos a esta restricción *bloqueo simétrico*.

De ahora en mas, sea $\mathsf{T} \in \mathsf{Tab}(\varphi)$, y sea Θ una rama de T. En lo que sigue, trabajaremos con *permutaciones en capas* tal como fueron definidas en las Definiciones 2.33 y 2.34.

Definición 5.4 (Distancia desde el prefijo raíz)**.** *Sea $\alpha \in \mathsf{PREF}$ un prefijo. Definimos a $\mathsf{depth}(\alpha)$ como la distancia desde el prefijo raíz al prefijo α, medida como el número de declaraciones de accesibilidad que deben ser atravesadas para alcanzar el prefijo α desde el prefijo raíz (en particular, si α es el prefijo de φ luego $\mathsf{depth}(\alpha) = 0$).*

Dada una fórmula prefijada $\alpha{:}\varphi$ y una permutación en capas $\bar{\sigma}$, definimos $\bar{\sigma}(\alpha{:}\varphi) = \alpha{:}\bar{\sigma}_{\mathsf{depth}(\alpha)+1}(\varphi)$. Finalmente, dada una fórmula modal φ, definimos $Vars(\varphi)$ como el conjunto de variables proposicionales que ocurren en φ; para S un conjunto de fórmulas modales, sea $Vars(S) = \bigcup_{\varphi \in S} Vars(\varphi)$.

Definición 5.5 (Bloqueo Simétrico)**.** *Sea $\bar{\sigma}$ una simetría en capas de φ, y sea Θ una rama en un tableau de φ. La regla $(\Diamond)$ no puede ser aplicada a $\alpha{:}\bar{\sigma}(\neg\Box\psi)$ en Θ si ya ha sido aplicada a $\alpha{:}\neg\Box\psi$ y $Vars(\bar{\sigma}(\neg\Box\psi)) \cap Vars(\Gamma(\alpha)) = \emptyset$, para $\Gamma(\alpha) = \{\psi \mid \alpha : \Box\psi \in \Theta\}$ el conjunto de $\Box$-fórmulas que ocurren en el prefijo α.*

Notar que el bloqueo simétrico es una condición dinámica: luego de ser bloqueada, una $\neg\Box$-formula puede ser agendada nuevamente para su expansión si la condición de bloqueo falla en una expansión de la rama. Esto puede suceder ya que el conjunto $\Gamma(\alpha)$ crece monotónicamente a medida que el tableau avanza. También Notar que esta restricción no afecta la terminación del tableau ni la correctitud del mismo, ya que estamos imponiendo aún más restricciones sobre un método que se sabe correcto, completo y que termina. Solo nos arriesgamos a perder completitud.

5.2.1 *Completitud*

Para probar la completitud de nuestro cálculo con bloqueo simétrico vamos a mostrar que podemos extender un modelo incompleto $\mathcal{M}^\Theta$, construido a partir de una rama abierta y saturada Θ, a un modelo completo, aún cuando el bloqueo simétrico es utilizado.

De ahora en más, consideramos que las ramas en $\mathsf{Tab}(\varphi)$ están saturadas usando bloqueo simétrico.

Definición 5.6. *Dada una rama abierta y saturada Θ del tableau $\mathsf{T} \in \mathsf{Tab}(\varphi)$, definimos el modelo $\mathcal{M}^\Theta = \langle W^\Theta, R^\Theta, V^\Theta \rangle$ como:*

$$\begin{aligned} W^\Theta &= \{\alpha \mid \alpha \text{ es un prefijo en } \Theta\} \\ R^\Theta &= \{(\alpha, \alpha') \mid \alpha, \alpha' \in W^\Theta \text{ y } \alpha R \alpha' \in \Theta\} \\ V^\Theta(\alpha) &= \{p \mid \alpha : p \in \Theta\}. \end{aligned}$$

Notar que $\mathcal{M}^\Theta$ siempre es un árbol. Dado un modelo arbolar y una permutación en capas, construimos un nuevo modelo de la siguiente forma.

Definición 5.7 ($\bar{\sigma}$-*imagen* de un modelo arbolar)**.** *Dado un modelo arbolar puntuado $\mathcal{M} = \langle w, W, R, V \rangle$ y una permutación en capas $\bar{\sigma}$. Sea $\mathsf{depth}(v)$ la distancia desde v a la*

*raíz w (en particular $\mathsf{depth}(w) = 0$). Definimos $\mathcal{M}_{\bar{\sigma}} = \langle w, W_{\bar{\sigma}}, R_{\bar{\sigma}}, V_{\bar{\sigma}} \rangle$ (la $\bar{\sigma}$-*imagen *de $\mathcal{M}$) como el modelo idéntico a $\mathcal{M}$ excepto que*

$$V_{\bar{\sigma}}(v) = \bar{\sigma}_{\mathsf{depth}(v)+1}(L_{V(v)}) \cap \mathsf{PROP},$$

donde $L_{V(v)}$ es el conjunto de literales completo y consistente generado por $V(v)$ (Definición 2.9).

Dado un modelo $\mathcal{M}$ y un elemento $w \in W$, $W[w]$ denota el conjunto de todos los elementos que son alcanzables desde w (a través de la clausura reflexo-transitiva de la relación de accesibilidad). Con $\mathcal{M}[w] = \langle w, W[w], R\restriction W[w], V\restriction W[w] \rangle$ denotamos el submodelo de $\mathcal{M}$ con raíz en w.

Dada una rama abierta y saturada Θ, sea Σ el conjunto de prefijos agregados a Θ por la aplicación de la regla $(\Diamond)$ a una $\neg\Box$-fórmula que tiene una $\neg\Box$-fórmula simétrica bloqueada usando bloqueo simétrico. Intuitivamente, Σ contiene las raíces de los submodelos que necesitan una contraparte simétrica cuando se complete el modelo $\mathcal{M}^{\Theta}$.

Sea $M[\Sigma] = \{\mathcal{M}^{\Theta}[\alpha] \mid \alpha \in \Sigma\}$. Este conjunto contiene los submodelos para los cuales necesitamos construir un submodelo simétrico. Con $M[\Sigma]_{\bar{\sigma}} = \{\mathcal{M}^{\Theta}[\alpha]_{\bar{\sigma}} \mid \mathcal{M}^{\Theta}[\alpha] \in M[\Sigma]\}$ denotamos al conjunto de las $\bar{\sigma}$-imágenes correspondientes al conjunto de submodelos $M[\Sigma]$. Intuitivamente, $M[\Sigma]_{\bar{\sigma}}$ es el conjunto de modelos que necesitamos "pegar" al modelo $\mathcal{M}^{\Theta}$ para completarlo.

Definición 5.8 (Extensión Simétrica)**.** *Dada una rama abierta y saturada Θ, un modelo $\mathcal{M}^{\Theta} = \langle W^{\Theta}, R^{\Theta}, V^{\Theta} \rangle$ y un conjunto de submodelos puntuados simétricos $M[\Sigma]_{\bar{\sigma}}$. Definimos la extensión simétrica de $\mathcal{M}^{\Theta}$ como el modelo $\mathcal{M}^{\Theta}_{\bar{\sigma}} = \langle W^{\Theta}_{\bar{\sigma}}, R^{\Theta}_{\bar{\sigma}}, V^{\Theta}_{\bar{\sigma}} \rangle$ donde:*

$$\begin{aligned} W^{\Theta}_{\bar{\sigma}} &= W^{\Theta} \uplus \biguplus W \\ R^{\Theta}_{\bar{\sigma}} &= R^{\Theta} \uplus \biguplus R \cup \{(\alpha, \tau_{\alpha'}) \mid (\alpha, \alpha') \in R^{\Theta}\} \\ V^{\Theta}_{\bar{\sigma}}(\alpha_i) &= V^{\Theta} \uplus \biguplus V \end{aligned}$$

para todo $\langle \tau_{\alpha'}, W, R, V \rangle \in M[\Sigma]_{\bar{\sigma}}$, donde $\tau_{\alpha'}$ es el elemento correspondiente a α' en la unión disjunta.

Notar que estamos *pegando* los submodelos simétricos al modelo original agregando un arco desde el elemento α a la raíz, $\tau_{\alpha'}$, del submodelo simétrico si existe un arco desde α a la raíz, α', del submodelo original. Para estar seguros que la construcción es correcta, debemos verificar que luego de agregar estos submodelos al modelo original, las $\Box$-fórmulas que valían en α, $\Gamma(\alpha)$, siguen valiendo.

El siguiente lema es clave para probar la completitud de nuestro cálculo. Primero recordamos el siguiente resultado.

Proposición 5.1. *Sea φ una fórmula modal y PROP un conjunto de variables proposicionales tal que $Vars(\varphi) \subseteq \mathsf{PROP}$. Sea $\mathcal{M}$ un modelo tal que $V : W \mapsto \mathcal{P}(\mathsf{PROP})$. Luego $\mathcal{M}, w \models \varphi$ sii $\mathcal{M}\restriction Vars(\varphi), w \models \varphi$.*

Lema 5.1. *Sea φ una fórmula en CNF modal, $\bar{\sigma}$ una simetría de φ y Θ una rama abierta y saturada de $\mathsf{T} \in \mathsf{Tab}(\varphi)$. Sea α un prefijo tal que $\alpha{:}\neg\Box\psi \in \Theta$ y sea $\alpha{:}\bar{\sigma}(\neg\Box\psi) \in \Theta$ una fórmula bloqueada. Luego $\bar{\sigma}(\psi) \bigwedge \Gamma(\alpha)$ es satisfacible.*

Demostración. Ver [Orbe, 2014]. □

Ahora podemos probar la correspondencia entre fórmulas en la rama Θ y su validez en la extensión simétrica de un modelo construido a partir de ella.

Lema 5.2. *Sea Θ una rama abierta y saturada de un tableau $\mathsf{T} \in \mathsf{Tab}(\varphi)$ y $\bar{\sigma}$ una simetría de φ. Para toda fórmula $\alpha : \psi \in \Theta$ tenemos que $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \alpha \models \psi$.*

Demostración. Ver [Orbe, 2014]. □

Teorema 5.2. *El cálculo de tableaux con bloqueo simétrico para la lógica modal básica es completo.*

Demostración. Sea Θ una rama abierta y saturada del tableau $\mathsf{T} \in \mathsf{Tab}(\varphi)$. Dado que $0 : \varphi \in \Theta$, por el Lema 5.2 tenemos que φ es satisfacible. □

5.3 EVALUACIÓN EXPERIMENTAL

A continuación presentamos resultados experimentales sobre como se comporta el bloqueo simétrico en conjuntos de fórmulas modales.

5.3.1 *Implementación*

Para verificar los efectos del bloqueo simétrico (SB) en un tableaux modal, lo implementamos en el prover `HTab` [Hoffmann and Areces, 2007]. La implementación es directa: cuando hay una $\neg\Box$-fórmula agendada para su expansión, el solver verifica si ya existe una fórmula simétrica que ya haya sido expandida. Si este es el caso, el solver bloquea la $\neg\Box$-fórmula y continua con la aplicación de las siguientes reglas. El solver solo verifica la condición de bloqueo si obtiene una rama abierta y saturada. Si la condición de bloqueo se mantiene para todas las fórmulas bloqueadas, el solver termina. De lo contrario reagenda las fórmulas, para las cuales la condición de bloqueo no se mantiene, para su subsiguiente expansión. La información de simetrías es provista como una entrada adicional junto con la fórmula.

5.3.2 *Resultados*

Nuestro conjunto de prueba incluye 954 fórmulas simétricas provenientes del Logics Workbench Benchmark for *K* (LWB_K) [Balsiger *et al.*, 2000] y de la librería QBF-LIB [Giunchiglia *et al.*, 2001]. Los problemas provenientes de la QBF-LIB son traducidos a la lógica modal básica usando la herramienta `qbf2ml`[1] usando la traducción *Collapse*1, una variante de la traducción de Ladner que genera fórmulas con menor profundidad modal y por ende más pequeñas. Todos los tests fueron ejecutados en un Intel Core i7 2,93GHz con 16GB of RAM con un timeout de 600 segundos.

La Tabla 5.1 presenta los resultados con y sin bloqueo simétrico (`HTab`+SB y `HTab`, respectivamente). Las columnas *#Suc* y *#To* representa el número de instancias para las cuales el solver pudo establecer su estatus y el número de instancias para las cuales el solver alcanzó el timeout respectivamente. Las columnas T_1 y T_2 son los tiempos totales (incluyendo el tiempo de computo de simetrías), en segundos, en

1 Descargar desde:`http://cs.famaf.unc.edu.ar/~ezequiel/resource/qbf2ml`

el conjunto completo de fórmulas, incluyendo y excluyendo los timeouts respectivamente. La tabla muestra que HTab+SB supera a HTab: HTab+SB requiere menos tiempo para resolver todas las instancias y es capaz de resolver 7 instancias más que HTab (HTab+SB es capaz de resolver 9 instancias que llegan al timeout con HTab, pero alcanza el timeout en otras 2 instancias que HTab es capaz de resolver). También muestra que un gran número de fórmulas alcanzaron el timeout. La mayor parte de estas fórmulas provienen de la QBF-LIB, que, debido a la traducción a la lógica modal básica, resultan en fórmulas modales extremadamente grandes.

Solver	$\#Suc$	$\#To$	T_1	T_2
HTab+SB	318	636	9657	391167
HTab	311	643	10634	396434

Tabla 5.1: Tiempos totales con (HTab+SB) y sin (HTab) bloqueo simétrico.

Status	$\#In$	$\#Trig$	B_1	B_2
Satisfacibles	157	73	6319	6278
No satisfacibles	163	79	1038	87

Tabla 5.2: Aplicaciones del bloqueo simétrico.

La Figura 5.2 presenta un gráfico de dispersión de los tiempos de ejecución para las 320 fórmulas que se pudieron resolver en al menos una de las configuraciones. El eje x da el tiempo de ejecución para HTab mientras que el eje y da el tiempo de ejecución de HTab+SB. Cada punto representa una instancia y sus coordenadas horizontal y vertical representan el tiempo necesario para resolverla en segundos. Puntos debajo de la diagonal representan instancias donde HTab+SB supera a HTab, mientras que los puntos sobre la diagonal representan instancias para las cuales HTab supera a HTab+SB. Puntos en los extremos superior y derecho representan timeouts. Notar que se utiliza una escala logarítmica, por lo cual una ganancia o una degradación más hacia arriba o hacia la derecha es exponencialmente más relevante. La figura muestra que HTab+SB supera a HTab. Aproximadamente la mitad de las instancias reportan una ganancia de performance, mientras que la otra mitad reporta una leve perdida de performance. Sin embargo, una mirada en profundidad a la Figura 5.2 nos muestra que la mayor parte de la ganancia se obtiene en instancias que no son triviales de resolver, y que obtenemos una ganancia de varios órdenes de magnitud para muchas instancias. Mientras que la degradación de performance se registra mayormente en instancias triviales (es decir, en instancias que requieren menos de un segundo para ser resueltas). En estos casos, la degradación se debe al procesamiento extra impuesto por el mecanismo de bloqueo en instancias que nunca disparan el bloqueo simétrico. Sin embargo, esta degradación es negligible para la mayor parte de las instancias.

Clasificamos las 320 instancias, que fueron resueltas en al menos una de las configuraciones, en instancias satisfacibles e instancias no satisfacibles para entender mejor el efecto del bloqueo simétrico. La Tabla 5.2 muestra información sobre la aplicación del bloqueo simétrico en cada categoría. Las columnas $\#In$ y $\#Trig$ representan el número de instancias en cada conjunto y el número de instancias para

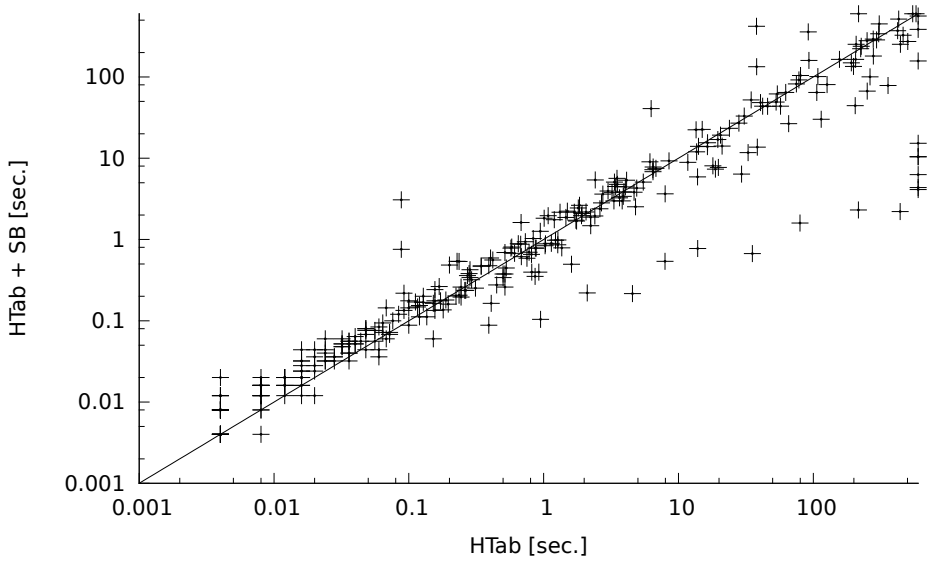


Figura 5.2: Performance de HTab vs. HTab+SB en todas las fórmulas.

las cuales el bloqueo simétrico se disparó al menos una vez respectivamente. Las columnas B_1 y B_2 representa el número de veces que el bloqueo simétrico es disparado y el número de veces que una fórmula que fue inicialmente bloqueada tuvo que ser reagendada debido a una violación de la condición de bloqueo, respectivamente. Para ambas categorías, observamos que el bloqueo simétrico se dispara en aproximadamente la mitad de las instancias (46 % para las instancias satisfacibles y 48 % para las instancias no satisfacibles). Para las instancias satisfacibles, el bloqueo simétrico se dispara muchas veces pero, en la mayor parte de los casos, la condición de bloqueo falla luego en la rama. Para el caso de instancias no satisfacibles, el bloqueo simétrico se dispara menos veces, pero en la mayoría de los casos la condición de bloqueo se mantiene (recordar que la condición de bloqueo solo se verifica si la rama esta saturada y abierta).

Las Figuras 5.3 y 5.4 muestran la misma información que la Figura 5.2 pero diferenciando entre fórmulas satisfacibles y no satisfacibles. La Figura 5.3 muestra que en fórmulas satisfacibles HTab+SB supera a HTab a pesar de que la mayor parte de las veces la condición de bloqueo falla luego en la rama. Esto se puede explicar por el hecho de que, en muchos casos donde la condición de bloqueo falla, retrasar el procesamiento de fórmulas simétricas puede ser beneficioso ya que la rama tiene más información disponible que puede evitar el branching o cerrar la rama más rápidamente. También observamos que la degradación en performance debido a este procesamiento extra es negligible.

La Figura 5.4 muestra que para fórmulas no satisfacibles, HTab+SB logra una importante mejora en algunas instancias. También se observa que HTab+SB resuelve 7 instancias más que HTab. La degradación también es más notoria en algunas instancias. Una posible explicación para esto es la siguiente: si la fórmula bloqueada no juega rol alguno en la no satisfacibilidad del problema, entonces el bloqueo evita trabajo innecesario, lo cual resulta en una ganancia de performance. Si al contrario, la fórmula bloqueada cumple un rol en la no satisfacibilidad del problema y su procesamiento se retrasa debido al bloqueo, el solver puede verse forzado a proce-

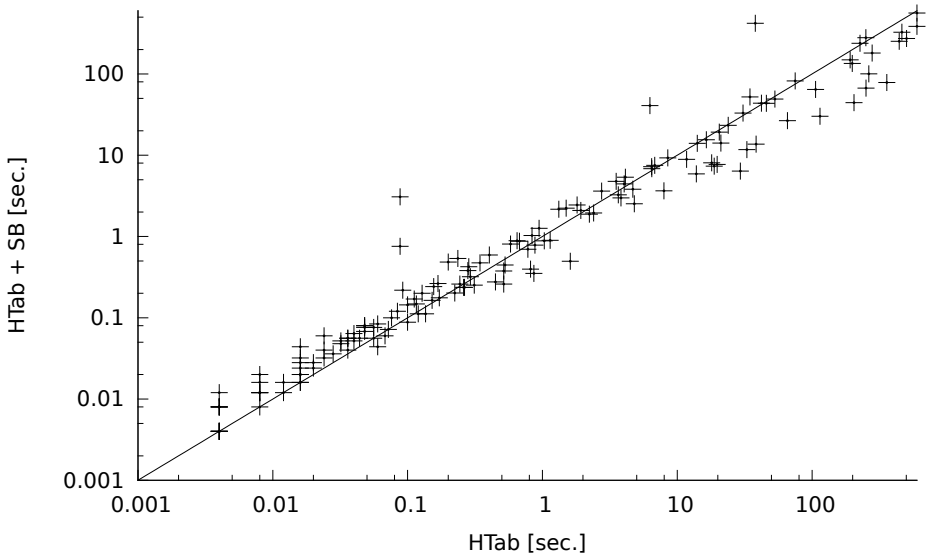


Figura 5.3: Performance de `HTab` vs. `HTab`+SB: Fórmulas Satisfacibles.

sar fórmulas que no serían procesadas de no haber sido bloqueada dicha fórmula, lo cual resulta en una pérdida de performance.

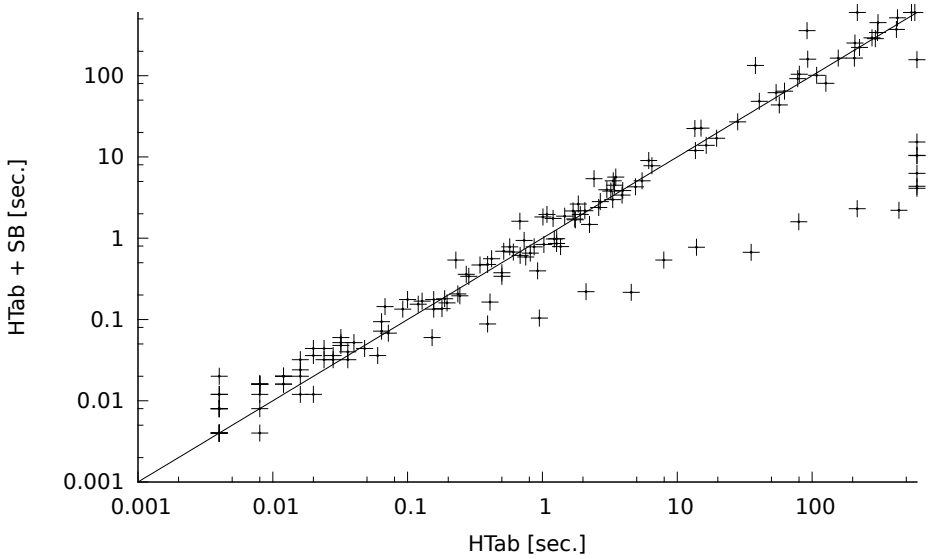


Figura 5.4: Performance de `HTab` vs. `HTab`+SB: Fórmulas no satisfacibles.

De los datos obtenidos en nuestros tests, queda claro que la efectividad del bloqueo simétrico depende fuertemente de la clase de problema. Para algunas clases de problemas, el bloqueo simétrico provee una gran ganancia en la performance (e. j., la clase `k_branch` del LWB_K). Por otra parte, en algunas clases altamente simétricas, el bloqueo simétrico no hace diferencia ya que nunca es disparado. En otras palabras, el bloqueo simétrico solo se ocupa de un subconjunto de todas las simetrías presentes en las fórmulas modales.

Parte III

CONCLUSIONES

6

CONCLUSIONES Y TRABAJO FUTURO

Es momento de resumir el trabajo que hemos realizado en esta tesis. Como ya se ha mencionado, nuestro objetivo fue investigar las simetrías en el contexto de las lógicas modales y SMT. Veamos ahora las contribuciones realizadas en cada uno de estos dominios.

6.1 SIMETRÍAS EN LAS LÓGICAS MODALES

El estudio de las simetrías en las lógicas modales ha sido el principal tema de este trabajo.

Comenzamos nuestro trabajo desarrollando los fundamentos teóricos necesarios para poder utilizar las simetrías de un problema modal y, para ello, probamos dos resultados fundamentales. Primero, probamos que las simetrías (*globales*) de una fórmula de la lógica modal básica particionan el espacio de modelos en clases de equivalencia de forma tal que cada clase de equivalencia contiene sólo modelos que satisfacen la fórmula o sólo modelos que no la satisfacen. Luego, probamos que las simetrías pueden ser utilizadas como un mecanismo de inferencia, y por lo tanto, pueden ser usadas para fortalecer los mecanismos de razonamiento existentes. Finalmente, extendimos estos resultados a un gran abanico de lógicas modales usando el marco provisto por los modelos modales coinductivos e introdujimos las *permutaciones en capas*, para aquellas lógicas con la propiedad de modelo arbolar, que nos permiten definir un noción de simetrías más flexible .

Luego, centramos nuestra atención en la detección de simetrías en fórmulas modales. Para ello, extendimos una técnica existente para la lógica proposicional que reduce el problema de detección de simetrías al problema de detectar automorfismos en un grafo construido a partir de la fórmula. Presentamos dos algoritmos de construcción, uno para detectar simetrías *globales* y otro para detectar simetrías *en capas*. Probamos que ambos algoritmos son correctos, y que, por lo tanto, todo automorfismo detectado se corresponde con una simetría de la fórmula a partir de la cual fue construido. Ya que ambos algoritmos fueron desarrollados en el marco de los modelos coinductivos, por lo cual se pueden considerar como algoritmos *modelo* de los cuales se pueden derivar implementaciones concretas para las distintas lógicas modales. A continuación implementamos estos algoritmos para la lógica modal básica y los evaluamos en diferentes conjuntos de fórmulas modales para determinar cuán simétricas son estas fórmulas y cuán difícil es detectar las simetrías en las mismas. Los resultados experimentales mostraron que las simetrías si existen en las fórmulas modales y que su presencia esta fuertemente relacionada con la codificación del problema. También indicaron que detectar las simetrías de un problema es un problema generalmente fácil, aún para fórmulas de gran tamaño.

Finalmente, pusimos las simetrías a trabajar, para lo cual implementamos un cálculo de tableaux que incorpora un mecanismo de bloqueo, llamado *bloqueo simétrico*, que usa la información de simetrías de una fórmula para restringir la apli-

cación de la regla ($\Diamond$). La idea es aplicar la regla ($\Diamond$) a una $\neg\Box$-fórmula solo si la regla no fue aplicada previamente a una $\neg\Box$-fórmula simétrica. Probamos que el cálculo con bloqueo simétrico es completo e implementamos dicho cálculo en el solver `HTab`. Luego lo evaluamos experimentalmente en diferentes conjuntos de fórmulas modales. Los resultados experimentales nos mostraron que usando el mecanismo de bloqueo es posible obtener importantes ganancias de performance en un gran número de fórmulas. Los datos también indicaron que el mecanismo de bloqueo no siempre se dispara, incluso en fórmulas altamente simétricas. Esto es un indicador de que nuevas técnicas, que utilicen información de simetrías, deben ser desarrolladas para aprovechar toda la información disponible.

Los resultados presentados en este trabajo, tanto teóricos como experimentales, deben ser considerados como el primer paso hacia un desarrollo más cabal del estudio de las simetrías en las lógicas modales, dado que existe mucho espacio para mejorar, tanto en el aspecto teórico como en el experimental.

Permítannos ahora describir las líneas de investigación a futuro que consideramos más interesantes.

En el aspecto teórico, debería ser posible desarrollar la teoría de simetrías para lógicas modales que no puedan ser representadas usando el marco de los modelos modales coinductivos y para lógicas que son variantes notacionales de las lógicas modales, e. j., Lógicas de Descripción.

En esta tesis, solo trabajamos con simetrías proposicionales (simetrías de átomos en el marco coinductivo) que permutan sólo literales proposicionales. Debería ser posible definir una noción de simetría que involucre literales modales. Notar que una permutación de literales proposicionales implica una permutación de literales modales. Sin embargo, debería ser posible encontrar literales modales simétricos que no sean implicados por literales proposicionales simétricos. Una posible dirección para resolver este tema sería considerar la *abstracción proposicional* de una fórmula modal, como se hace en SMT, y hacer la detección de simetrías en ella.

En el lado experimental, hay todavía mucho trabajo por realizar en lo que respecta a utilizar las simetrías en el razonamiento modal. Un camino posible es investigar como desarrollar predicados de ruptura de simetrías para fórmulas modales como es hecho para fórmulas proposicionales.

Otra línea de investigación interesante, es la utilización de simetrías en cálculos modales basados en resolución. Clave para esto es el uso de las simetrías como mecanismo de inferencia. En particular, una aplicación directa de la regla de simetría introducida en [Krishnamurthy, 1985] debería ser posible.

Una alternativa que también merece atención es la utilización de simetrías como mecanismo de simplificación, es decir, utilizar las simetrías para simplificar la fórmula, removiendo subfórmulas simétricas. Esta idea es similar en espíritu a lo que se hace con el bloqueo simétrico, pero ahora realizado directamente sobre la fórmula antes de darla al solver.

Finalmente, una advertencia: debido a la gran variedad de lógicas modales, parece difícil encontrar un único método que sirva para usar las simetrías en todas las lógicas modales. Para obtener mejores resultados, es probable que se deban desarrollar métodos específicos para cada lógica modal, e inclusive para cada tipo de problemas.

6.2 SIMETRÍAS EN SATISFACIBILIDAD MODULO TEORÍAS

En el contexto de Satisfacibilidad Modulo Teorías (SMT), en esta tesis nos enfocamos en el desarrollo de una nueva técnica de detección de simetrías. Para ello, extendimos la técnica basada en grafos usada en lógica proposicional a SMT, y presentamos un algoritmo de construcción que crea los grafos a partir de una fórmula SMT. Luego implementamos este algoritmo en la herramienta SyMT y lo evaluamos en la SMT-LIB. Los resultados experimentales mostraron que usando esta técnica de detección es posible detectar simetrías que no se pueden detectar con otras técnicas existentes.

Como herramienta de inspección, SyMT, puede ayudar a los usuarios a identificar las simetrías en una fórmula para luego eliminarlas, y así mejorar la codificación de los problemas.

Todavía existe mucho espacio para mejorar la técnica de detección y la herramienta en si misma. Como trabajo futuro sería interesante adaptar el algoritmo de Schreir-Sims a nuestro uso para mejorar la eficiencia de este método sistemático de identificación de subgrupos. En muchos casos, los grupos de simetrías contienen subgrupos que son grupos de permutación completos para cierto subconjunto de símbolos. Mejorar la presentación en esos casos proveerá de mejor información a los usuarios de SyMT. Finalmente, la herramienta actualmente solo detecta simetrías que involucran permutaciones de símbolos no interpretados, esto, por supuesto, no cubre todos los casos de simetrías posibles. Por ejemplo, las fórmulas `queens` tienen simetrías que involucran razonamiento aritmético.

De forma similar a lo que ocurre en las lógicas modales, existe mucho trabajo por realizar en lo que respecta a aprovechar la información de simetrías en un solver SMT. Primero, el desarrollo de predicados de ruptura de simetrías debería ser investigado. Resultados preliminares muestran que, al igual que en el caso proposicional, la ruptura automática de simetrías en SMT no es la panacea. Creemos que en la mayor parte de los casos, es necesario el conocimiento del usuario para diseñar buenos predicados de ruptura de simetrías, y que el diseño de estas fórmulas dependerá de la clase de problemas y de la teoría en cuestión.

Otra posible línea de investigación, es investigar los efectos de utilizar las simetrías como mecanismo de inferencia en un solver SMT, de forma similar a lo que se hace en [Benhamou *et al.*, 2010]. En esta línea, el uso de las simetrías dentro de los solvers específicos de cada teoría también debería ser considerado.

BIBLIOGRAFÍA

[Aloul *et al.*, 2003a] F. Aloul, I. Markov, and K. Sakallah. Shatter: efficient symmetry-breaking for boolean satisfiability. In *Design Automation Conference.*, pages 836–839. IEEE, 2003. (Citado en páginas 6 and 7.)

[Aloul *et al.*, 2003b] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(9):1117–1137, 2003. (Citado en páginas 6, 7, 26, and 39.)

[Aloul *et al.*, 2006] F. Aloul, K. Sakallah, and I. Markov. Efficient symmetry breaking for boolean satisfiability. *Computers, IEEE Transactions on*, 55(5):549–558, 2006. (Citado en páginas 6 and 7.)

[Arai and Urquhart, 2000] N. Arai and A. Urquhart. Local symmetries in propositional logic. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 40–51, 2000. (Citado en página 5.)

[Areces and Gorín, 2010] C. Areces and D. Gorín. Coinductive models and normal forms for modal logics (or how we learned to stop worrying and love coinduction). *Journal of Applied Logic*, 8(4):305–318, 2010. (Citado en páginas v and 15.)

[Areces and Heguiabehere, 2003] C. Areces and J. Heguiabehere. hGen: A Random CNF Formula Generator for Hybrid Languages. In *Methods for Modalities 3*, Nancy, France, 2003. (Citado en página 30.)

[Areces and Orbe, 2013] C. Areces and E. Orbe. Dealing with symmetries in modal tableaux. In D. Galmiche and D. Larchey-Wendling, editors, *TABLEAUX*, volume 8123 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 2013. (Citado en páginas 25 and 49.)

[Areces *et al.*, 2000] C. Areces, R. Gennari, J. Heguiabehere, and M. de Rijke. Tree-based heuristics in modal theorem proving. In *Proceedings of ECAI'2000*, pages 199–203, Berlin, Germany, 2000. (Citado en página 21.)

[Areces *et al.*, 2012] C. Areces, G. Hoffmann, and E. Orbe. Symmetries in modal logics. In D. Kesner and P. Viana, editors, *LSFA*, volume 113 of *EPTCS*, pages 27–44, 2012. (Citado en páginas 9 and 25.)

[Areces *et al.*, 2013] C. Areces, D. Deharbe, P. Fontaine, and E. Orbe. Symt: finding symmetries in smt formulas. In *11th International Workshop on Satisfiability Modulo Theories (SMT 2013)*, Helsinki, Finland, 07/2013 2013. (Citado en página 39.)

[Audemard *et al.*, 2002] G. Audemard, A. Cimatti, A. Kornilowicz, and R. Sebastiani. Bounded model checking for timed systems. In *Formal Techniques for Networked and Distributed Sytems (FORTE 2002)*, pages 243–259. Springer, 2002. (Citado en páginas 8 and 39.)

[Audemard *et al.*, 2004] G. Audemard, B. Mazure, and L. Sais. Dealing with symmetries in quantified boolean formulas. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 257–262, 2004. (Citado en páginas v and 7.)

[Audemard *et al.*, 2006] G. Audemard, B. Benhamou, and L. Henocque. Predicting and detecting symmetries in fol finite model search. *Journal of Automated Reasoning*, 36(3):177–212, 2006. (Citado en página v.)

[Audemard *et al.*, 2007a] G. Audemard, S. Jabbour, and L. Sais. Efficient symmetry breaking predicates for quantified boolean formulae. In *Proceedings of SymCon-Symmetry in Constraitns-CP workshop*. Citeseer, 2007. (Citado en página 8.)

[Audemard *et al.*, 2007b] G. Audemard, S. Jabbour, and L. Sais. Symmetry breaking in quantified boolean formulae. In *IJCAI*, pages 2262–2267, 2007. (Citado en páginas v and 8.)

[Audemard, 2002] G. Audemard. Reasoning by symmetry and function ordering in finite model generation. In *Automated Deduction (CADE-18)*, pages 226–240, 2002. (Citado en página v.)

[Balsiger *et al.*, 2000] P. Balsiger, A. Heuerding, and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. *Journal of Automated Reasoning*, 24(3):297–317, 2000. (Citado en páginas 30 and 53.)

[Barrett *et al.*, 2009] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. *Satisfiability Modulo Theories*, chapter 26, pages 825–885. Volume 185 of Biere et al. [2009], February 2009. (Citado en página 39.)

[Barrett *et al.*, 2010a] C. Barrett, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010. (Citado en página 46.)

[Barrett *et al.*, 2010b] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010. (Citado en página 40.)

[Benhamou and Sais, 1992] B. Benhamou and L. Sais. Theoretical study of symmetries in propositional calculus and applications. In *Automated Deduction (CADE-11)*, pages 281–294, 1992. (Citado en páginas 4, 5, and 7.)

[Benhamou and Sais, 1994] B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning*, 12(1):89–102, 1994. (Citado en páginas 5 and 7.)

[Benhamou *et al.*, 2010] B. Benhamou, T. Nabhani, R. Ostrowski, and M. Saidi. Enhancing clause learning by symmetry in SAT solvers. In *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 329–335, 2010. (Citado en páginas 7, 15, and 61.)

[Beth, 1959] E. Beth. *The foundations of mathematics: a study in the philosophy of sciences*. North-Holland Amsterdam, 1959. (Citado en página 49.)

[Biere *et al.*, 2009] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009. (Citado en páginas 64 and 69.)

[Blackburn *et al.*, 2001] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001. (Citado en páginas 9, 11, 17, and 20.)

[Blackburn *et al.*, 2006] P. Blackburn, J. van Benthem, and F. Wolter. *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*. Elsevier Science Inc., New York, NY, USA, 2006. (Citado en páginas 9 and 49.)

[Bošnački *et al.*, 2002] D. Bošnački, D. Dams, and L. Holenderski. Symmetric spin. *International Journal on Software Tools for Technology Transfer*, 4(1):92–106, 2002. (Citado en página 7.)

[Bouton *et al.*, 2009] T. Bouton, D. De Oliveira, D. Déharbe, and P. Fontaine. verit: an open, trustable and efficient smt-solver. In *Automated Deduction (CADE-22)*, pages 151–156. Springer, 2009. (Citado en páginas 8 and 39.)

[Brading and Castellani, 2013] K. Brading and E. Castellani. Symmetry and symmetry breaking. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. The Stanford Encyclopedia of Philosophy, spring 2013 edition, 2013. (Citado en página 3.)

[Brown *et al.*, 1989] C. Brown, L. Finkelstein, and P. Purdom Jr. Backtrack searching in the presence of symmetry. In *Applied algebra, algebraic algorithms and error-correcting codes*, pages 99–110. Springer, 1989. (Citado en páginas 5 and 7.)

[Clarke *et al.*, 1996] E. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1-2):77–104, 1996. (Citado en páginas 7 and 8.)

[Cohen *et al.*, 2005] D. Cohen, P. Jeavons, C.r Jefferson, K. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems. In *Principles and Practice of Constraint Programming*, pages 17–31. Springer, 2005. (Citado en página 7.)

[Cohen *et al.*, 2009] M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A symmetry reduction technique for model checking temporal-epistemic logic. In *IJCAI*, volume 9, pages 721–726, 2009. (Citado en página 8.)

[Crawford *et al.*, 1996] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proceedings of KR 1996*, pages 148–159, 1996. (Citado en páginas 6 and 7.)

[Crawford, 1992] J. Crawford. A theoretical analysis of reasoning by symmetry in first-order logic. In *Proceedings of AAAI Workshop on Tractable Reasoning*, pages 17–22, San Jose, CA, 1992. (Citado en páginas 5, 6, and 7.)

[Cubadda, 1988] C. Cubadda. *Variantes de l'algorithmes de sl-résolution avec retenue d'informations*. PhD thesis, GIA Luminy (Marseille), 1988. (Citado en página 5.)

[D'Agostino, 1999] M. D'Agostino. *Handbook of tableau methods*. Springer, 1999. (Citado en página 49.)

[Darga *et al.*, 2004] P. Darga, M. Liffiton, K. Sakallah, and I. Markov. Exploiting structure in symmetry detection for CNF. In *Design Automation Conference.*, pages 530–534, 2004. (Citado en páginas 25 and 35.)

[Darga *et al.*, 2008] P. Darga, K. Sakallah, and I. Markov. Faster symmetry discovery using sparsity of symmetries. In *Proceedings of the 45th annual Design Automation Conference*, pages 149–154. ACM, 2008. (Citado en páginas 25 and 35.)

[Darvas, 2007] G. Darvas. *Symmetry. Cultural-historical and ontological aspects of science-arts relations. The natural and man-made world in an interdisciplinary approach.* Basel: Birkhäuser, 2007. (Citado en página 3.)

[Déharbe *et al.*, 2011] D. Déharbe, P. Fontaine, S. Merz, and B. Woltzenlogel Paleo. Exploiting symmetry in smt problems. In *Automated Deduction (CADE-23)*, volume 6803 of *Lecture Notes in Computer Science*, pages 222–236. Springer Berlin Heidelberg, 2011. (Citado en páginas 8 and 39.)

[Donaldson and Miller, 2005] A. Donaldson and A. Miller. Automatic symmetry detection for model checking using computational group theory. In *Formal Methods*, pages 481–496. Springer, 2005. (Citado en página 8.)

[Donaldson, 2007] A. Donaldson. *Automatic techniques for detecting and exploiting symmetry in model checking*. PhD thesis, University of Glasgow, 2007. (Citado en página 8.)

[Een and Sörensson, 2003] N. Een and N. Sörensson. An extensible sat-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2003. (Citado en página 7.)

[Fitting, 1972] M. Fitting. Tableau methods of proof for modal logics. *Notre Dame Journal of Formal Logic*, 13(2):237–247, 1972. (Citado en página 49.)

[Fitting, 1983] M. Fitting. *Proof methods for modal and intuitionistic logics*. D. Reidel (Dordrecht, Holland and Boston, USA and Hingham, MA), 1983. (Citado en página 49.)

[Fortin, 1996] S. Fortin. The graph isomorphism problem. Technical report, Technical Report 96-20, University of Alberta, Edomonton, Alberta, Canada, 1996. (Citado en página v.)

[Fox and Long, 1999] M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *IJCAI*, volume 99, pages 956–961, 1999. (Citado en página 7.)

[Fox and Long, 2002] M. Fox and D. Long. Extending the exploitation of symmetries in planning. In *AIPS*, pages 83–91, 2002. (Citado en página 7.)

[Fraleigh and Katz, 2003] J. Fraleigh and V. Katz. *A first course in abstract algebra*. Addison-Wesley world student series. Addison-Wesley, 2003. (Citado en página 12.)

[Gent *et al.*, 2006] I. P Gent, K. Petrie, and J. Puget. Symmetry in constraint programming. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 10. Elsevier, 2006. (Citado en páginas v and 7.)

[Giunchiglia *et al.*, 2001] E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001. `http://www.qbflib.org`. (Citado en páginas 30 and 53.)

[Goré, 1999] R. Goré. Tableau methods for modal and temporal logics. In Marcello D'Agostino, DovM. Gabbay, Reiner Hähnle, and Joachim Posegga, editors, *Handbook of Tableau Methods*, pages 297–396. Springer Netherlands, 1999. (Citado en página 49.)

[Harrison, 2009] J. Harrison. Without loss of generality. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *Lecture Notes in Computer Science*, pages 43–59, Munich, Germany, 2009. Springer-Verlag. (Citado en página 3.)

[Hoffmann and Areces, 2007] G. Hoffmann and C. Areces. Htab: A terminating tableaux system for hybrid logic. In *Proceedings of Methods for Modalities 5*, November 2007. (Citado en página 53.)

[Hon and Goldstein, 2008] G. Hon and B. Goldstein. *From Summetria to Symmetry: The Making of a Revolutionary Scientific Concept*, volume 20. Springer Netherlands, 2008. (Citado en página 3.)

[Horrocks *et al.*, 2000] I. Horrocks, P. Patel-Schneider, and R. Sebastiani. An analysis of empirical testing for modal decision procedures. *Logic Journal of IGPL*, 8(3):293–323, 2000. (Citado en página 29.)

[Ip and Dill, 1996] C. Ip and D. Dill. Better verification through symmetry. *Formal methods in system design*, 9(1-2):41–75, 1996. (Citado en página 7.)

[Junttila and Kaski, 2007] T. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2007. (Citado en páginas 25, 30, and 35.)

[Katebi *et al.*, 2012] H. Katebi, K. Sakallah, and I. Markov. Conflict anticipation in the search for graph automorphisms. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 243–257. Springer, 2012. (Citado en páginas 25, 35, and 45.)

[Kowalski and Kuehner, 1972] R. Kowalski and D. Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2(3):227–260, 1972. (Citado en página 5.)

[Kripke, 1959] S. Kripke. A Completeness Theorem in Modal Logic. *Journal of Symbolic Logic*, 24(1):1–14, 1959. (Citado en página 10.)

[Kripke, 1963] S. Kripke. Semantical considerations on modal logic. *Acta Philos. Fennica*, 16:83–94, 1963. (Citado en página 10.)

[Krishnamurthy, 1985] B. Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22(3):253–275, 1985. (Citado en páginas 4, 5, and 60.)

[Ladner, 1977] R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM journal on computing*, 6(3):467–480, 1977. (Citado en página 30.)

[Lis, 1960] Z. Lis. Wynikanie semantyczne a wynikanie formalne. *Studia Logica*, 10(1):39–54, 1960. (Citado en página 49.)

[Margot, 2002] F. Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94(1):71–90, 2002. (Citado en página 7.)

[Margot, 2003] F. Margot. Exploiting orbits in symmetric ilp. *Mathematical Programming*, 98(1-3):3–21, 2003. (Citado en página 7.)

[Massacci, 1994] F. Massacci. Strongly analytic tableaux for normal modal logics. In *Automated Deduction (CADE-12)*, pages 723–737. Springer, 1994. (Citado en página 49.)

[McKay, 2007] B. McKay. Nauty userś guide (version 2.4). *Computer Science Dept., Australian National University*, 2007. (Citado en página 25.)

[Miller *et al.*, 2006] A. Miller, A. Donaldson, and M. Calder. Symmetry in temporal logic model checking. *ACM Comput. Surv.*, 38(3), September 2006. (Citado en página 8.)

[Moskewicz *et al.*, 2001] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001. (Citado en página 6.)

[Orbe, 2014] E. Orbe. Symmetries in automated reasoning: The case of modal logics and satisfiability modulo theories. 2014. . Disponible en `http://cs.famaf.unc.edu.ar/~ezequiel/publication/symmetries-automated-reasoning`. (Citado en páginas 14, 15, 18, 19, 22, 27, 29, 30, 52, and 53.)

[Oxusoff, 1989] L. Oxusoff. *L'évaluation sémantique en calcul propositionnel*. PhD thesis, Université Aix-Marseille 2, 1989. (Citado en página 5.)

[Patel-Schneider and Sebastiani, 2003] P. Patel-Schneider and R. Sebastiani. A new general method to generate random modal formulae for testing decision procedures. *Journal of Artificial Intelligence Research*, 18:351–389, 2003. (Citado en página 10.)

[Pólya and Read, 1987] G. Pólya and R. Read. *Combinatorial enumeration of groups, graphs, and chemical compounds*. Springer-Verlag New York, Inc., 1987. (Citado en página 6.)

[Rehn and Schürmann, 2010] T. Rehn and A. Schürmann. C++ tools for exploiting polyhedral symmetries. In Komei Fukuda, Jorisvander Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software*, volume 6327 of *LNCS*, pages 295–298. Springer, 2010. (Citado en página 45.)

[Roe, 2006] K. Roe. The heuristic theorem prover: Yet another smt modulo theorem prover. In *Computer Aided Verification*, pages 467–470. Springer, 2006. (Citado en páginas 8 and 39.)

[Sabharwal, 2005] A. Sabharwal. Symchaff: A structure-aware satisfiability solver. In *AAAI*, volume 5, pages 467–474, 2005. (Citado en páginas 6 and 7.)

[Sakallah, 2009] K. Sakallah. *Symmetry and Satisfiability*, chapter 10, pages 289–338. Volume 185 of Biere et al. [2009], February 2009. (Citado en páginas v, 4, and 25.)

[Sebastiani, 2007] R. Sebastiani. Lazy satisfiability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:141–224, 2007. (Citado en página 39.)

[Seress, 1997] A. Seress. An introduction to computational group theory. *Notices of the AMS*, 44:671–679, 1997. (Citado en páginas 6 and 12.)

[Seress, 2003] A. Seress. *Permutation Group Algorithms*. Cambridge Tracts in Mathematics. Cambridge University Press, 2003. (Citado en página 45.)

[Sistla *et al.*, 2000] A. Sistla, V. Gyuris, and E. Emerson. Smc: a symmetry-based model checker for verification of safety and liveness properties. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(2):133–166, 2000. (Citado en página 7.)

[Smullyan, 1968] R. Smullyan. *First-order logic*. Springer-Verlag (Berlin and New York etc), 1968. (Citado en página 49.)

[Urquhart, 1999] A. Urquhart. The symmetry rule in propositional logic. *Discrete Applied Mathematics*, 96–97(0):177 – 193, 1999. (Citado en página 4.)

[van Benthem, 1977] J. van Benthem. *Modal Correspondence Theory*. PhD thesis, Universiteit van Amsterdam, 1977. (Citado en página 11.)

[van Benthem, 1983] J. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, 1983. (Citado en página 11.)