SIMETRÍAS EN LÓGICAS DE DESCRIPCIÓN

GIOVANNI RESCIA

Trabajo Especial de la Licenciatura en Ciencias de la Computación

Facultad de Matemática, Astronomía, Física y Computación
Universidad Nacional de Córdoba

Director: Dr. Ezequiel Orbe

Marzo 2017

A mi familia, que me ha apoyado y guiado
en cada decisión que he tomado.

AGRADECIMIENTOS

Agradezco principalmente a mi director, el Dr. Ezequiel Orbe, por haber depositado su confianza en mí para realizar este trabajo, por haberme ayudado a mejorarlo en cada iteración y por hacerse siempre un tiempo para despejar mis dudas.

Al Dr. Carlos Areces, quien ha colaborado durante el desarrollo de esta tesis aportando su vasta experiencia en los temas tratados aquí. Su contribución ha sido fundamental para este trabajo.

Al Dr. Miguel Pagano, porque sus palabras de aliento hicieron posible que este trabajo de tesis lleve mi nombre.

A mis amigos con quienes compartí mi vida académica a través de incontables horas de estudio, litros de mate, líneas de código y vueltas al Paseo del Buen Pastor.

A Alejandro, por siempre motivarme e inspirarme a apuntar alto.

A Facundo y Ernesto, porque en cada juntada se celebra la amistad.

Córdoba, 19/05/2017

RESUMEN

En esta tesis trabajaremos con el concepto de simetría en el contexto del razonamiento automático. Si podemos identificar las simetrías de un problema, podríamos utilizarlas para reducir la dificultad del razonamiento analizando en detalle sólo uno de los casos simétricos y luego generalizar el resultado a los demás. Esto es exactamente lo que tratamos de hacer cuando utilizamos las simetrías de un problema en el contexto de razonamiento automático: la presencia de simetrías en subespacios que no contienen soluciones pueden ser de utilidad para nuestro algoritmo ya que podemos guiarlo para que busque soluciones en otras partes del espacio de búsqueda. En particular, nos enfocaremos en la detección de simetrías en lógicas de descripción.

En [45] se da un marco teórico para la detección y el uso de simetrías en diversas lógicas modales y en [2] se presenta una conexión entre lógicas modales y lógicas de descripción. Aprovecharemos estos resultados para desarrollar un nexo sintáctico y semántico entre ambas lógicas y así trasladar todos los resultados teóricos necesarios para la detección de simetrías obtenidos para lógicas modales hacia el caso de las lógicas de descripción.

También definiremos un algoritmo para construir un grafo coloreado a partir de una fórmula CNF modal. Se ha demostrado que dicho algoritmo genera un grafo tal que todos sus automorfismos tienen una correspondencia uno-a-uno con las simetrías de la fórmula, por lo que trasladaremos el problema de detección de simetrías al de detección de automorfismos de grafos.

Con respecto al marco práctico, describiremos la herramienta implementada para la detección de simetrías en lógicas de descripción y estudiaremos cada uno de sus módulos a través de un simple caso de uso.

Presentaremos también resultados de un estudio empírico al probar nuestra herramienta con ontologías obtenidas de la web y utilizadas en razonadores, y analizaremos dichos resultados: tiempos de ejecución, tamaños de las ontologías, tamaños de los grafos y cantidad de simetrías obtenidas.

ABSTRACT

In this thesis we will work with the concept of symmetry, in the domain of automatic reasoning. If one is able to identify the symmetries of a problem, then that information could be used to reduce the difficulty of analyzing in detail only one of these simmetric cases and generalizing after. This is exactly what we try to do when we use the symmetries of a problem in automatic reasoning: the presence of symmetries in subspaces that do not contain any solutions can be used by our algorithm, guiding it to search for solutions somewhere else. In particular, we will focus on detecting symmetries in description logics.

In [45] the detection and application of symmetries in different types of modal logics is broadly studied, also, in [2] a link between modal logics and description logics is presented. We will exploit this results to set a sintactic and semantic relation among both logics and be able to transfer every theoretical result needed to detect symmetries in modal logics towards the case of description logics.

We will also define an algorithm for constructing a graph given a formula in modal CNF. It has been proved that this algorithm generates a graph such that its automorphism have a one-to-one mapping with the formula's symmetries.

With respect to the practical part, we will describe the system implemented for detecting symmetries in description logics and we will analyze each one of its modules through a simple use case.

Results obtained by an empirical study from testing our system on ontologies acquired from the web and used in reasoners, are also presented and analized in detail: computation times, ontologies size, graphs size and number of symmetries obtained.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

1 INTRODUCCIÓN

1.1 QUÉ VAMOS A HACER EN ESTA TESIS

Supongamos que estamos frente a una computadora que contiene una base de datos a la cual podemos hacer consultas. Ahora supongamos que la computadora conoce la respuesta para alguna consulta de tipo A, pero nosotros queremos hacerle alguna otra consulta de tipo B. Finalmente, supongamos que contamos con un algoritmo que puede decidir si dos consultas son *similares* cuando se toman en cuenta ciertos aspectos. Entonces, nuestra computadora podría valerse de esta información y reutilizar una respuesta similar a la de a A para contestar B.

Intuitivamente, ese será el foco de trabajo de esta tesis: detectar *similitudes* en consultas que se realizan sobre bases de datos; donde estas similitudes serán introducidas formalmente bajo el nombre de *simetrías*. Las bases de datos particulares con las que trabajaremos son conocidas como *bases de conocimiento* ($\mathcal{KB}$), y estarán especificadas formalmente en el lenguaje que proveen las *Lógicas de Descripción* ($\mathcal{DL}$), mientras que el lenguaje para implementar estas bases de conocimiento en una computadora, será el lenguaje *OWL*, un lenguaje inspirado en las lógicas de descripción.

1.2 CAPTURANDO CONOCIMIENTO

La historia de la Inteligencia Artificial (IA) ha mostrado que el *conocimiento* es crítico a la hora de desarrollar sistemas de IA. En muchos casos, a la hora de resolver una tarea, contar con *mejor* conocimiento puede ser más importante que contar con mejores algoritmos.

Para poder tener sistemas verdaderamente inteligentes, el conocimiento necesita ser capturado, procesado, reusado y comunicado. Las ontologías dan soporte a todas estas tareas.

1.2.1 *Ontologías*

El término *ontología*[1] tiene sus raíces en la rama de la filosofía que está enfocada en el estudio del ser o la existencia.

En el contexto de las ciencias de la computación, ontología refiere a un término técnico que denota un artefacto que es *diseñado* para un propósito: permitir el modelado de conocimiento sobre cierto dominio específico, tanto real como ficticio.

Este término ha sido adoptado por investigadores en el área de IA, quienes argumentaron que se podrían crear nuevas ontologías vistas como modelos computacionales que permitirían cierto tipo de razonamiento automático [32]. En la década de 1980 la comunidad de IA comenzó a usar el término ontología para referirse

1 http://dle.rae.es/?w=ontología

tanto a una teoría de un mundo modelado (por ejemplo, Naïve Physics [32]), como a un componente de los sistemas de conocimiento.

A comienzos de 1990, se identificó a la ontología como un componente estándar en todos los sistemas de conocimiento [44]. En [30] se define ontología como una *especificación explícita de una conceptualización*, que refiere a *los objetos, conceptos y otras entidades que son parte de algún área de interés y las relaciones entre éstos*. Mientras que los términos *especificación* y *conceptualización* pueden prestarse para diferentes interpretaciones, los puntos esenciales en la definición de ontología son:

- Una ontología define (especifica) los conceptos, relaciones, y otras distinciones que son relevantes para modelar el dominio.
- La especificación toma la forma de las definiciones del vocabulario (clases, relaciones, etc.), que proveen un significado para el mismo junto con restricciones formales para su uso coherente.

1.2.2 *Representando ontologías*

Distintos abordajes para la representación de ontologías se desarrollaron en los años setenta, los cuales pueden ser clasificados en dos categorías: representaciones no basadas en lógica y formalismos basados en lógica.

1.2.2.1 *Sistemas no basados en lógica*

Entre los sistemas más populares no basados en lógica encontramos dos: las *redes semánticas* y las *estructuras (frames)*.

En las redes semánticas [49] el conocimiento se representa en la forma de un grafo dirigido con etiquetas: cada nodo es asociado con un concepto y los arcos representan relaciones de algún tipo entre conceptos. Las redes semánticas fueron creadas en un intento para expresar *interlingua*, un lenguaje común que sería usado para traducir entre varios lenguajes.

Un ejemplo típico es *WordNet* [61], el cual describe relaciones entre palabras del inglés, que usando lenguaje natural puede relacionar, por ejemplo, la palabra *alimento* con *nutriente.*

En el caso de las estructuras [42], el objetivo se centra en lograr conocimiento estructurado basándose en la capacidad de expresar relaciones entre estructuras. Una estructura representa cierto concepto y es caracterizada por un número de atributos, donde los valores de éstos pueden ser elementos de algún dominio concreto (e.g., enteros, *strings* [conjunto de caractéres]) o identificadores de otras estructuras. Un ejemplo del uso de modelos basados en estructuras es *OKBC* [17], el cual define una interfaz para acceder a sistemas de representación de conocimiento. OKBC define la mayoría de los conceptos que son propios en los sistemas de estructuras y bases de datos relacionales.

Si bien las redes semánticas y las estructuras contienen diferencias muy marcadas, tanto en las motivaciones que llevaron a su desarrollo como en sus prestaciones, comparten una base común. De hecho, nos podemos referir a ambas como *estructuras de red*, donde la estructura de cada red apunta a representar conjuntos de individuos y cómo éstos se relacionan entre sí.

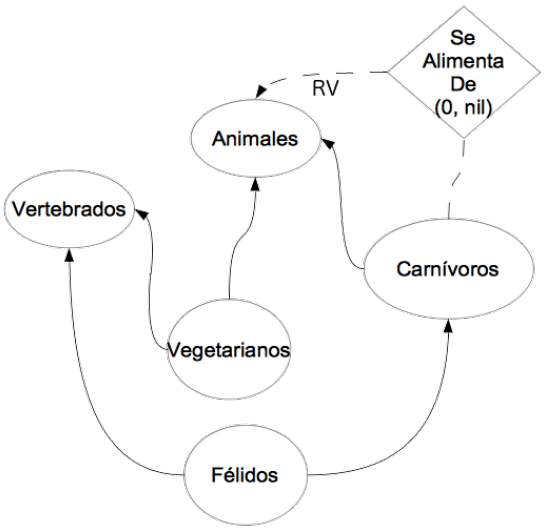

Figura 1.1: Representación de relaciones entre ciertos animales con una red genérica

Ejemplo 1.1. Para dar un poco de intuición acerca de las ideas claves en los sistemas de representación basados en redes, hablaremos en términos de una red genérica, sin hacer referencia a algún sistema en concreto. La Figura 1.1 representa conocimiento modelando una pequeña parte del mundo animal utilizando una red genérica. La estructura utilizada también es referida como una *terminología*, y representa la generalización/especificación de los conceptos involucrados. Por ejemplo, el vínculo entre ***Félidos*** y ***Carnívoros*** indica que todos los félidos son carnívoros; este tipo de relación también es llamada una relación *IS-A* (en español, "es-un/es-una"). La relación IS-A define una jerarquía entre los conceptos que implica la herencia de atributos: si un concepto A es más específico que otro concepto B, entonces el concepto A hereda las propiedades del concepto B. Por ejemplo, si decimos que los carnívoros tienen ciertos órganos particulares que usan para capturar a su presa, entonces los félidos también cuentan con dichos órganos. Siguiendo con el ejemplo de la Figura 1.1, vemos que el concepto Carnívoro está conectado con el nodo ***seAlimentaDe***, el cual es una propiedad también llamada *rol*. Este rol cuenta con una *restrición de valor*, denotado por la etiqueta *RV*, que expresa una limitación en el rango de tipos de objetos que pueden relacionarse con el rol. En nuestro ejemplo, el rol tiene una restricción en cuanto a la cardinalidad, expresada como (0, *nil*), donde el primer número es el límite inferior y el segundo número el límite superior (*nil* representa infinito). Entonces, podría leerse como *un carnívoro es un animal que se alimenta de (0 o más) animales.* Un detalle importante en estas estructuras son las relaciones implícitas que se encuentran entre los conceptos. Por ejemplo, como todos los carnívoros son animales, y un félido es un carnívoro, entonces un félido es un animal. Encontrar este tipo de relaciones implícitas es el objetivo de un sistema de representación de conocimiento.

Una limitación de este tipo de representación es que a medida que las relaciones entre los conceptos se vuelven más complejas, aumenta la dificultad para dar caracterizaciones precisas sobre qué tipos de relaciones pueden ser computadas y para garantizar la correctitud y completitud del sistema que razona sobre estas estructuras.

1.2.2.2 *Sistemas basados en lógica*

Usando las ideas que hemos visto hasta el momento, varios sistemas fueron implementados y utilizados en diversas aplicaciones. Como resultado, surgió la necesidad de una caracterización precisa del significado de las estructuras usadas en las representaciones y del conjunto de inferencias que podían obtenerse de dichas estructuras.

Una caracterización precisa del significado de una estructura de red puede obtenerse definiendo un lenguaje para los elementos de la misma (sintaxis) y proveyendo una interpretación para estos símbolos (semántica). Si bien la sintaxis puede variar bajo ciertos aspectos (e.g., dominio del problema), la semántica suele estar dada en términos de modelos. En [11, 12] se encuentran algunas de las primeras referencias que abordan sobre el tema de contar con una semántica formal para estas redes semánticas.

En particular, el trabajo realizado por Brachman llevó al desarrollo de KL-ONE [15], uno de los primeros sistemas de conocimiento, el cual contaba con cierto tipo de semántica lógica. Muchos sistemas evolucionaron a partir de KL-ONE, como ser KRYPTON [13], LOOM [40], CLASSIC [10], BACK [48] y KRISS [5]. Todo este trabajo también dio sus frutos en investigaciones teóricas sobre lo que se llamó *lenguaje de concepto* o *terminológico*, y que hoy se conoce como *lógicas de descripción.*

Las lógicas de descripción son una familia de lógicas que proveen de una sintaxis y semántica formal para la representación de conocimiento.

Para la sintaxis se introdujo un lenguaje abstracto [43], definido a partir de dos alfabetos disjuntos que son utilizados para denotar *conceptos atómicos* (símbolos unarios de predicado) y *roles atómicos* (símbolos binarios de predicado) para expresar relaciones entre los conceptos.

Las fórmulas se construyen a partir de los símbolos básicos usando distintos operadores. Por ejemplo, si C y D son conceptos, los símbolos que utilizamos para representar la *intersección de conceptos*, y la *unión entre conceptos*, son $\sqcap$ y $\sqcup$, respectivamente, y pueden ser utilizados, por ejemplo como $C \sqcap D$ y $C \sqcup D$.

Con respecto a la semántica, un concepto es interpretado como un conjunto de individuos mientras que los roles son interpretados como relaciones binarias. Los conceptos atómicos son interpretados como subconjuntos del dominio de interpretación, mientras que la semántica de los otros constructores es especificada definiendo el conjunto de *individuos* (donde un *individuo* es un elemento particular de un concepto, e.g., *W.A. Mozart* podría ser un individuo de la clase *Músico*) denotado por cada constructor. Por ejemplo, el concepto $C \sqcup D$ es el conjunto de individuos obtenido uniendo los conjuntos de individuos denotados por C y por D, respectivamente, o sea que se podría interpretar al operador $\sqcup$ como el operador unión de conjuntos. Mientras que $C \sqcap D$ restringe el conjunto de individuos bajo consideración a aquellos que pertenecen tanto a C como a D, es decir, el operador $\sqcap$ se interpreta como el operador intersección de conjuntos, pero sobre conceptos. Por último, las relaciones son conjuntos de pares de individuos.

Ejemplo 1.2. Supongamos que *Persona, Hombre* y *Mujer* son conceptos atómicos, *Hércules* y *Alcmena* son individuos y que *hijoDe* es un rol atómico. Usando los operadores *intersección, unión* y *complemento* de conceptos, interpretados como operaciones entre conjuntos, podemos describir nuevos conceptos, como por ejemplo

las personas que no son mujeres, los individuos que son mujeres u hombres o incluso que *Alcmena es madre de Hércules*, denotados por las siguientes expresiones:

$$Persona \sqcap \neg Mujer$$
$$Mujer \sqcup Hombre$$
$$hijoDe(Hércules, Alcmena)$$

Ya dimos una introducción de sistemas que tienen una semántica formal y bien definida en los cuales se puede razonar, ahora bien, ¿cómo explotamos esta capacidad para inferir información implícita? Para poder realizar inferencias necesitamos primero representar la información de un determinado dominio en el formato adecuado. Una de las fuentes de información principales hoy en día es la web. La *web semántica* es una propuesta de formalización de la información existente en la web.

1.2.3 *La web semántica*

La *World Wide Web*, o *WWW*, es una librería de documentos entrelazados que son transferidos entre computadoras y presentados a las personas. Actualmente la WWW contiene muchísima información y conocimiento, pero las máquinas por lo general solo transportan y presentan el contenido de los documentos que describen al conocimiento. Son las personas las encargadas de conectar todas las fuentes de información relevante e interpretarlas.

La web semántica [60], un esfuerzo en colaboración liderado por el Consorcio de la WWW (W3C), tiene como objetivo mejorar la actual web para que las computadoras sean capaces de procesar la información presentada en la WWW, interpretarla y conectarla, para ayudar a las personas a encontrar el conocimiento requerido. De la misma manera que la WWW, la web semántica es un sistema distribuido de hipertexto, que está pensado para formar un gran sistema de conocimiento distribuido. El foco de la web semántica es compartir datos, en vez de documentos. En otras palabras, es un proyecto que busca proveer *una estructura común que permita a la información ser compartida y reusada a través de los límites de la aplicación, el trabajo y la comunidad.*

La arquitectura sobre la cual la web semántica está construida cuenta con varias capas, pero dado el foco de esta tesis, no analizaremos todas ellas y solo presentaremos las capas relevantes que están dadas por la Figura 1.2.

Las capas *Unicode* y *URI* son los cimientos de esta arquitectura. Unicode [58] es un estándar para la codificación de conjuntos internacionales de caractéres y permite usar cualquier lenguaje humano (para escritura y lectura) en la web. URI [59] (*Uniform Resource Identifier*) es un string con un formato estandarizado que permite identificar recursos de forma única (e.g., documentos).

La capa *XML* (*Extensible Markup Language*) [62] junto con las definiciones de *XML namespace* y *XML schema* se aseguran de que haya una sintaxis común utilizada en la web semántica. XML es un lenguaje de marcas (etiquetas) de propósito general para documentos que contienen información estructurada.

La capa *RDF* (*Resource Description Framework*) [50] provee un formato básico de representación de la información para la web semántica. Este es un sistema para

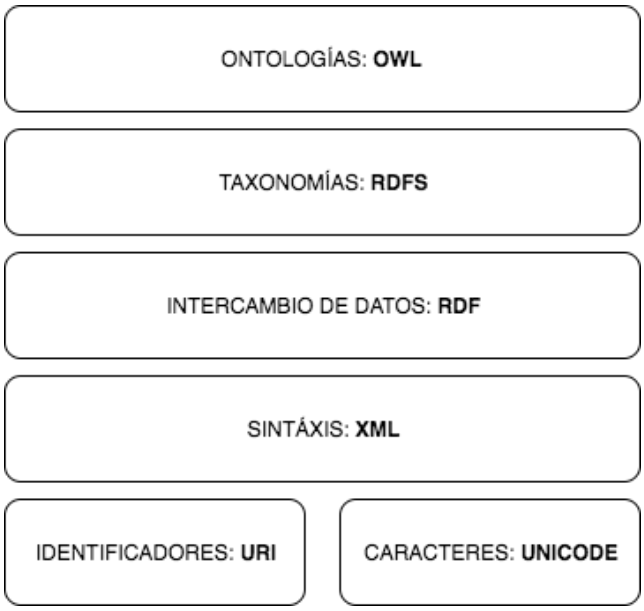

Figura 1.2: Algunas capas que componen la web semántica

representar información, acerca de los recursos, en forma de grafo. Está basado en triplas *sujeto-predicado-objeto* que conforman el grafo de datos. Todos los datos en la web semántica usan RDF como el principal lenguaje de representación. Debido a que RDF estaba pensando principalmente para representar metadatos sobre recursos de la WWW, fue construido con recursos de URI.

La capa RDF *Schema* (*RDFS*) [51] permite descripciones estandarizadas de taxonomías y otros constructores *ontológicos*, dentro de RDF. RDFS puede ser utilizado para describir taxonomías de clases y propiedades y así crear ontologías pequeñas.

Finalmente, la capa *OWL* (*Ontology Web Language*) [22] permite crear ontologías más detalladas. OWL es un lenguaje derivado de las lógicas de descripción y ofrece más construcciones sobre RDFS. Está sintácticamente embebido dentro de RDF, así que como RDFS, provee un vocabulario estandarizado adicional. Como dijimos, el lenguaje OWL para ontologías, está basado en las lógicas de descripción (de hecho, al sublenguaje de OWL que está basado en lógicas de descripción se lo refiere como OWL_{DL}). Esto tiene dos consecuencias inmediatas: primero, OWL cuenta con una semántica formal; y segundo, le aporta poder de razonamiento a la web semántica.

En esta tesis, OWL es de suma importancia, ya que es el lenguaje en el cual están expresadas todas las ontologías que utilizaremos.

1.3 SIMETRÍAS EN RAZONAMIENTO AUTOMÁTICO

Cuando pensamos en el concepto de simetría, quizá lo primero que nos viene a la cabeza puede ser un polígono regular, un número capicúa o incluso una imagen del Test de Rorschach, como la que se muestra en la Figura 1.3.

Es decir pensamos en algo que se puede *partir sobre un eje*, y repite el mismo patrón, quizá con direcciones diferentes, en ambas partes.

El concepto de simetría tiene muchos usos. No sólo podemos estudiar las propiedades simétricas de los objetos que ya hemos mencionado (figuras, números, etc.)

Figura 1.3: Lámina 8 del Test de Rorschach

para comprender su comportamiento o revelar algún patrón, 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*.

La intuición que subyace a los argumentos basados en simetría es que en el curso de una prueba matemática, a menudo se utiliza un elemento arbitrario de un conjunto como representante de este conjunto, siempre que sus elementos sean simétricos, de modo que los argumentos subsiguientes se aplican en forma similar a los demás elementos del conjunto. Este principio nos permite reconocer que una tautología (o sea, una fórmula que es siempre verdadera) permanece invariante bajo ciertas permutaciones de nombres de variables y se vale de esta información para evitar repetir derivaciones de fórmulas intermedias que son meramente variantes permutacionales la una de la otra.

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

En dicho contexto, la presencia de simetrías en subespacios que no contienen soluciones pueden ser de utilidad para nuestro algoritmo ya que podemos guiarlo para que busque soluciones en otras partes del espacio de búsqueda.

Dentro del contexto de razonamiento automático, definiremos como *simetría* de un problema a una permutación de sus variables que preserve su estructura (su forma sintáctica) y por lo tanto, su conjunto de soluciones (modelos).

El uso de simetrías en el contexto de razonamiento automático fue ampliamente estudiado.

En [39] se presentan dos reglas de inferencia para un cálculo de resolución: la regla de *simetría global* y la regla de *simetría local*. Si Γ es un conjunto de cláusulas proposicionales y σ es una permutación 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 $\sigma(\Gamma) = \Gamma$. Por otra parte, la regla de simetría local nos permitirá inferir $\sigma(C)$ siempre que $\sigma(A)$ sea parte de Γ, para cada cláusula A usada en la derivación de C.

En [39] se muestra la utilidad de las reglas de simetría para algunos problemas pero no se presenta un algoritmo para realizar la detección de simetrías. Tampoco se discute en detalle cómo los procedimientos de resolución deben utilizar las reglas de simetría. El primer intento para explotar la presencia de simetrías en un algoritmo de búsqueda se discute en [16].

En [7, 8] un algoritmo de detección de simetrías y un algoritmo de búsqueda para explotar simetrías, son presentados. La detección de simetrías es realizada directamente sobre la fórmula de entrada. Una vez que las simetrías son detectadas, se utilizan en dos algoritmos de búsqueda: el algoritmo SLRI [21] y el algoritmo de evaluación semántica [46].

En [1, 20] se presenta un enfoque diferente para la detección de simetrías. 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, un subgrupo del grupo de simetrías de la fórmula es detectado mediante la detección del grupo de automorfismos del grafo resultante.

En el contexto de las lógicas modales (las cuales son una extensión de la lógica proposicional, que incluye operadores que expresan una *modalidad*, donde los operadores modales son expresiones que *califican* las sentencias), el estudio de simetrías fue abordado en el área de verificación de modelos (o *model checking*) para la lógica LTL [18, 23, 24, 41] y la lógica temporal-epistémica [19].

De gran importancia para esta tesis, es el trabajo realizado en [45] sobre simetrías en lógicas modales y satisfacibilidad módulo teorías (SMT).

En particular, en [45] se da un marco teórico para el uso de simetrías en diversas lógicas modales y se propone un algoritmo de detección basado en el enfoque propuesto en [20].

En esta tesis trasladaremos resultados teóricos y herramientas desarrolladas en [45] hacia las lógicas de descripción. Es natural preguntarse, ¿cómo es posible trasladar tales resultados de una lógica a otra? La conexión entre las dos lógicas es presentada en [55] y estudiada con mayor detalle en [2]; para el tipo de lógica modal que se trabajó en [45] y los tipos de lógicas de descripción con los que trabajeremos aquí, veremos que existe una traducción sintáctica y que los modelos son los mismos para ambas lógicas.

1.4 LA ESTRUCTURA DE ESTA TESIS

En el Capítulo 2 damos una definición formal de la sintaxis y semántica para las lógicas de descripción, introducimos las nociones básicas de la web semántica y mostramos la conexión entre las lógicas de descripción y OWL.

Las lógicas modales son introducidas en el capítulo Capítulo 3 junto a su sintaxis y semántica. Un resultado teórico importante es presentado aquí también: la función de traducción Ψ, que permite traducir fórmulas escritas en lenguaje OWL a su fórmula modal equivalente.

Damos un algoritmo para detectar simetrías en lógicas de descripción en el Capítulo 4, junto con el Teorema de satisfacibilidad para lógicas de descripción con simetrías.

En el Capítulo 5 describimos las herramientas implementadas para la detección de simetrías en lógicas de descripción.

Las herramientas presentadas en el Capítulo 5 son puestas a prueba con ontologías reales en el Capítulo 6, con un análisis de los resultados obtenidos.

Por último, en el Capítulo 7 damos un resumen de lo hecho, junto con sus limitaciones y propuestas de trabajo futuro.

2

LENGUAJES PARA LA REPRESENTACIÓN DE CONOCIMIENTO

En este capítulo presentaremos en profundidad a las lógicas de descripción, su sintaxis y semántica; y su relación con OWL.

2.1 LÓGICAS DE DESCRIPCIÓN

Las lógicas de descripción son una familia de lógicas que se utilizan para representar conocimiento de manera formal [43]. Una de sus características principales, es que pueden verse como fragmentos decidibles de la lógica de primer orden [43].

A su vez, resultan muy útiles a la hora de representar bases de conocimiento donde el objetivo es poder representar y extraer información. Las bases de conocimiento expresadas por las lógicas de descripción diferencian entre el conocimiento *intencional* (conocimiento general acerca del dominio del problema) y conocimiento *extensional* (específico a un problema en particular). Dichos conocimientos están capturados por dos componentes, la *TBox* y la *ABox*.

La TBox contiene conocimiento intencional en forma de una *terminología* (de ahí el término *Terminological Box*), y es construida a través de declaraciones que describen propiedades generales sobre los conceptos.

La ABox, es la encargada del conocimiento extensional, o aserciones (de ahí *Assertional Box*). Este conocimiento es específico para individuos del dominio.

De ahora en más, consideraremos a una base de conocimiento como una tupla $\langle TBox, ABox \rangle$.

Definiremos luego la sintaxis y semántica de estas lógicas de manera precisa, primero comencemos con algunos conceptos básicos.

El lenguaje de una lógica de descripción se define en dos niveles. En el primer nivel, se definen los operadores que pueden utilizarse para la definición de conceptos (lo que se llama el lenguaje de conceptos, CON). Y en el segundo nivel se definen los operadores que se pueden utilizar para la creación de las fórmulas que corresponden a la TBox y a la ABox.

Usualmente, los operadores que se pueden utilizar en las TBox yABox son comunes a todas las lógicas de descripción, las cuales se diferencian solo en qué operadores se permiten en sus lenguajes de conceptos.

Como ejemplo, definamos el lenguaje de concepto de la lógica de descripción llamada $\mathcal{ALC}$, una de las más utilizadas.

Definición 2.1. *(Sintaxis de* $\mathsf{CON}_{\mathcal{ALC}}$*). Sea* ATOM *un conjunto de* conceptos atómicos, ROL *un conjunto de* roles atómicos. *El lenguaje de conceptos de* $\mathcal{ALC}$, $\mathsf{CON}_{\mathcal{ALC}}$, *se define como*

$$\mathsf{CON}_{\mathcal{ALC}} ::= A \mid \top \mid \bot \mid \neg A \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C,$$

con $A \in \mathsf{ATOM}$, $R \in \mathsf{ROL}$ *y* $C, D \in \mathsf{CON}_{\mathcal{ALC}}$.

Los conceptos de la Definición 2.1 pueden usarse en las TBox y ABox de la siguiente manera.

Definición 2.2. *(TBox y ABox). Sea* $\mathcal{L}$ *una lógica de descripción. Una* TBox *de* $\mathcal{L}$ *es un conjunto finito de fórmulas de la forma*

$$C_1 \equiv C_2$$
$$C_1 \sqsubseteq C_2,$$

donde C_1 *y* C_2 *son fórmulas del lenguaje de concepto de* $\mathcal{L}$. $C_1 \equiv C_2$ *expresa que los conceptos son equivalentes, mientras que* $C_1 \sqsubseteq C_2$ *indica que todos los individuos que conforman* C_1 *también son parte del concepto* C_2.

Sea IND *un conjunto de* individuos. *Una* ABox *de* $\mathcal{L}$ *es un conjunto finito de fórmulas de la forma*

$$C(a)$$
$$R(a, b),$$

donde $a, b \in \mathsf{IND}$. $C(a)$ *define que el individuo a es un elemento particular del concepto C. Y* $R(a, b)$ *indica que los individuos a y b están relacionados mediante la relación R.*

Veamos algunos ejemplos sobre construcciones de TBox y ABox para comprender en detalle qué significan las construcciones de la Definición 2.2.

Ejemplo 2.1. La forma más básica de declaración en una TBox es la *definición* de concepto. Por ejemplo, podríamos definir el concepto de *mujer* como *las personas que no son hombres*

$$Mujer \equiv Persona \sqcap \neg Hombre$$

Tal declaración es interpretada como una equivalencia lógica, que provee condiciones necesarias y suficientes para clasificar individuos como mujeres.

La terminología es constituida por un conjunto de definiciones de conceptos. En particular, la tarea principal de construir una terminología es la *clasificación*, que refiere a colocar una nueva expresión de concepto en su lugar correcto con respecto a la jerarquía de conceptos visto como taxonomía. La clasificación puede lograrse verificando las relaciones de inclusión entre todos los conceptos junto con la nueva expresión de conceptos. Las expresiones de tipo

$$Hombre \sqsubseteq Persona$$

son llamadas *axiomas de inclusión generales.* Y en este caso, expresa que *todos los hombres son personas*.

Ejemplo 2.2. Como mencionamos, la ABox contiene conocimiento extensional acerca del dominio de interés, i.e., aserciones acerca de los individuos. Veamos como ejemplo las siguientes aserciones, en las cuales los individuos son *Julieta, Islandia, Reikavik y Batman*

$$Mujer \sqcap Persona(Julieta)$$
$$tieneCapital(Islandia, Reikavik)$$
$$Superhéroe(Batman)$$

La primera indica que el individuo *Julieta* es una persona que es mujer (este tipo de aserciones son usualmente llamadas *aserciones sobre conceptos*), mientras que la segunda expresa que la capital de *Islandia* es *Reikiavik* (a este tipo de aserciones también se las conoce como *aserciones sobre roles*); y la última que *Batman* es un *Superhéroe*. La tarea de razonamiento básica en una ABox es *comprobación de instancias*, la cual verifica si algún individuo en particular es una instancia (o pertenece) a cierto concepto.

Es importante resaltar que si en una base de conocimientos además de conceptos también contamos con instancias particulares de los elementos de esos conceptos, o sea individuos, la complejidad de razonamiento desde un punto de vista computacional es más costosa [25].

Ahora daremos semántica al lenguaje de concepto de $\mathcal{ALC}$, dado en la Definición 2.1, y a las construcciones de la TBox y ABox, expresadas en la Definición 2.2.

Definición 2.3. *(Semántica del lenguaje de conceptos). Sea $\mathcal{M} = \langle \mathcal{I}, \Delta \rangle$ un modelo, donde Δ es un conjunto no vacío e $\mathcal{I}$ es una* función de interpretación *tal que $\mathcal{I}(A) \subseteq \Delta$ para $A \in \mathsf{ATOM}$ y $\mathcal{I}(R) \subseteq \Delta \times \Delta$ para $R \in \mathsf{ROL}$. Definamos recursivamente en términos de conjuntos qué representa cada uno de los constructores restantes de $\mathsf{CON}_{\mathcal{ALC}}$*

$$\begin{aligned}
\mathcal{I}(\top) &\doteq \Delta \\
\mathcal{I}(\bot) &\doteq \emptyset \\
\mathcal{I}(\neg A) &\doteq \Delta \setminus \mathcal{I}(A) \\
\mathcal{I}(C \sqcap D) &\doteq \mathcal{I}(C) \cap \mathcal{I}(D) \\
\mathcal{I}(C \sqcup D) &\doteq \mathcal{I}(C) \cup \mathcal{I}(D) \\
\mathcal{I}(\forall R.C) &\doteq \{x \in \Delta \mid \forall y \in \Delta, (x,y) \in \mathcal{I}(R) \implies y \in \mathcal{I}(C)\} \\
\mathcal{I}(\exists R.C) &\doteq \{x \in \Delta \mid \exists y \in \Delta, (x,y) \in \mathcal{I}(R) \wedge y \in \mathcal{I}(C)\}
\end{aligned}$$

Definición 2.4. *(Semántica de las construcciones de TBox y ABox). Sea $\mathcal{M} = \langle \mathcal{I}, \Delta \rangle$ un modelo, donde Δ es un conjunto no vacío, y donde $\mathcal{I}$ también interpreta los elementos de* IND*: $\mathcal{I}(a) \in \Delta$, $a \in \mathsf{IND}$. Sean C y D conceptos, R una relación. Entonces tenemos que*

$$\begin{aligned}
\mathcal{I}(C(a)) &\doteq \top \textit{ sii } \mathcal{I}(a) \in \mathcal{I}(C) \\
\mathcal{I}(R(a,b)) &\doteq \top \textit{ sii } (\mathcal{I}(a), \mathcal{I}(b)) \in \mathcal{I}(R) \\
\mathcal{I}(C \sqsubseteq D) &\doteq \top \textit{ sii } \mathcal{I}(C) \subseteq \mathcal{I}(D) \\
\mathcal{I}(C \equiv D) &\doteq \top \textit{ sii } \mathcal{I}(C) = \mathcal{I}(D)
\end{aligned}$$

2.1.1 *Extrayendo información*

Hasta ahora, sólo hemos descrito cómo se puede representar conocimiento en una base de conocimiento. Es momento de describir cómo extraemos información de éstas. La principal tarea de extracción de información es la inferencia sobre el conocimiento existente en la base de conocimiento.

En cuanto a la inferencia, la operación básica sobre expresiones entre conceptos en lógicas de descripción es la *subsunción* (inclusión o inferencia), escrita por lo general como $\mathcal{KB} \models C \sqsubseteq D$, dada alguna base de conocimientos $\mathcal{KB}$. Determinar la inclusión refiere al problema de comprobar si el concepto denotado por D (*subsumidor*) es considerado más general que el concepto denotado por C (*subsumido*). En otras palabras, comprueba si el concepto C siempre denota un subconjunto del conjunto denotado por D, en todo modelo de la $\mathcal{KB}$. Por ejemplo, uno podría querer saber si para alguna base de datos $\mathcal{KB}$ en particular es verdad que $Persona \sqsubseteq Mujer$.

Otro tipo de inferencia sobre expresiones de conceptos es la de *satisfacibilidad*, que es el problema de comprobar si una expresión no denota necesariamente el concepto vacío. De hecho, la satisfacibilidad de conceptos es un caso especial de inclusión con el conjunto subsumidor siendo el conjunto vacío, significando que el concepto no es satisfacible.

2.1.2 *Satisfacibilidad*

Definamos a continuación cuándo consideraremos que una base de conocimiento es verdadera en un modelo, y cuándo las fórmulas de una TBox y una ABox son consecuencias lógicas de alguna base de conocimiento.

Definición 2.5. *Sean, $\mathcal{M} = \langle \Delta, \mathcal{I} \rangle$ un modelo, con Δ un conjunto no vacío e $\mathcal{I}$ una función de interpretación, y sea $\mathcal{KB} = \langle TBox, ABox \rangle$ una base de conocimiento, diremos que $\mathcal{M} \models \mathcal{KB}$ si y solo si para todo*

$$
\begin{array}{lll}
C_i \sqsubseteq C_j \in TBox & vale & \mathcal{I}(C_i \sqsubseteq C_j) = \top \\
C \equiv D \in TBox & vale & \mathcal{I}(C \equiv D) = \top \\
C(a) \in ABox & vale & \mathcal{I}(C(a)) = \top \\
R(a,b) \in ABox & vale & \mathcal{I}(R(a,b)) = \top
\end{array}
$$

En tal caso diremos que $\mathcal{M}$ es un modelo de $\mathcal{KB}$.

Por último, ya estamos en condiciones de definir cuándo una expresión en $\mathcal{DL}$ es consecuencia lógica de una $\mathcal{KB}$.

Definición 2.6. *Sea $\mathcal{M} = \langle \Delta, \mathcal{I} \rangle$ un modelo, $\mathcal{KB}$ una base conocimiento, C, C_i, C_j conceptos, a y b individuos y R una relación. Entonces tenemos que*

$$
\begin{array}{lll}
\mathcal{KB} \models C_i \sqsubseteq C_j & sii & \forall \mathcal{M}.(\mathcal{M} \models \mathcal{KB} \implies \mathcal{I}(C_i \sqsubseteq C_j) = \top) \\
\mathcal{KB} \models C_i \equiv C_j & sii & \forall \mathcal{M}.(\mathcal{M} \models \mathcal{KB} \implies \mathcal{I}(C_i \equiv C_j) = \top) \\
\mathcal{KB} \models C(a) & sii & \forall \mathcal{M}.(\mathcal{M} \models \mathcal{KB} \implies \mathcal{I}(C(a)) = \top) \\
\mathcal{KB} \models R(a,b) & sii & \forall \mathcal{M}.(\mathcal{M} \models \mathcal{KB} \implies \mathcal{I}(R(a,b)) = \top)
\end{array}
$$

2.1.3 *Complejidad de lógicas de descripción*

En [14], se argumenta que hay un balance (*tradeoff*, en inglés) entre la expresividad de un lenguaje de representación y la dificultad de hacer razonamientos sobre las representaciones construidas usando dicho lenguaje. En particular, se muestra que diferentes combinaciones de constructores pueden dar lugar a lenguajes con diferentes propiedades computacionales.

Dado que los constructores usados en la TBox y la ABox son usualmente los mismos en las distintas $\mathcal{DL}$, la complejidad de razonar en una lógica de descripción particular viene dada por la expresividad del lenguaje de conceptos. En la Definición 2.1 vimos el lenguaje de conceptos de $\mathcal{ALC}$. Otras lógicas de descripción permiten otras construcciones a la hora de definir un concepto, brindando mayor o menor expresividad y, por consiguiente, otra complejidad a la hora de razonar.

Como ejemplo, podemos definir la lógica $\mathcal{ALCQ}$, extendiendo $\mathcal{ALC}$ con los operadores de restricción sobre cardinalidad $\geq nR.C$ y $\leq nR.C$:

$$\mathsf{CON}_{\mathcal{ALCQ}} ::= \mathsf{CON}_{\mathcal{ALC}} \mid \leq nR.C \mid \geq nR.C,$$

con su semántica definida como:

$$\begin{aligned} \mathcal{I}(\geq nR.C) &\doteq \{x \in \Delta \mid |\{y \in \Delta \mid (x,y) \in \mathcal{I}(R),\ y \in \mathcal{I}(C)\}| \geq n\} \\ \mathcal{I}(\leq nR.C) &\doteq \{x \in \Delta \mid |\{y \in \Delta \mid (x,y) \in \mathcal{I}(R),\ y \in \mathcal{I}(C)\}| \leq n\} \end{aligned}$$

Así como introducimos un nuevo operador a $\mathcal{ALC}$ para obtener $\mathcal{ALCQ}$, podemos seguir modificando el lenguaje de concepto para obtener nuevas lógicas de descripción. Algunos ejemplos de las lógicas de descripción clásicas son:

- $\mathcal{AL}$, por *Attribute Language*, es una versión menos expresiva de $\mathcal{ALC}$, pues no cuenta con el complemento sobre conceptos.
- $\mathcal{SHIF}$ se obtiene extendiendo $\mathcal{ALC}$ con transitividad sobre roles, jerarquías sobre roles, roles inversos y propiedades funcionales.
- $\mathcal{SHOIN}$ es el resultado de agregar nominales y restricciones de cardinalidad a $\mathcal{SHIF}$.

Estos lenguajes se definen en detalle en [36].

Ahora que contamos con una noción sobre qué cosas puede expresar en cada lógica de descripción, veamos cuánto cuesta razonar sobre cada una de éstas, teniendo en cuenta que por lo general a mayor poder expresivo, mayor será el costo de razonamiento. La complejidad, en estos casos está dada por la inclusión de conceptos, i.e., por preguntar si $\mathcal{KB} \models C \sqsubseteq D$, para alguna $\mathcal{KB}$. En el Cuadro 2.1 listamos algunos lenguajes junto a su complejidad computacional[1], analizando por separado el caso en que $\mathcal{KB} = \emptyset$ y el caso en que $\mathcal{KB} \neq \emptyset$, ya que la complejidad en cada uno de los casos difiere.

1 http://www.cs.man.ac.uk/~ezolin/dl/

$\mathcal{DL}$	COMPLEJIDAD COMPUTACIONAL DE INCLUSIONES	
	Caso $\mathcal{KB} = \emptyset$	*Caso* $\mathcal{KB} \neq \emptyset$
$\mathcal{AL}$	*PTime*	*PSpace*
$\mathcal{ALC}$	*PSpace*	*ExpTime*
$\mathcal{ALCQ}$	*PSpace*	*ExpTime*
$\mathcal{SHIF}$	*ExpTime*	*ExpTime*
$\mathcal{SHOIN}$	*NExpTime*	*NExpTime*

Cuadro 2.1: Lógicas de descripción y su complejidad

2.2 ONTOLOGY WEB LANGUAGE (OWL)

Repasemos un poco ahora sobre ontologías, OWL, y su relación sintáctica y semántica con las lógicas de descripción.

2.2.1 *¿Qué es OWL?*

En el Capítulo 1 mencionamos que OWL [22] es lenguaje en el cual se definen las ontologías para la web semántica. Más precisamente, OWL es un lenguaje de ontologías, producido por el *W3C Web Ontology Working Group*, que fue diseñado para representar categorías de objetos y cómo éstos están relacionados.

Es importante resaltar que OWL no fue diseñado de manera aislada, sino que contó con muchas influencias. Analicemos un poco la historia de OWL.

2.2.2 *¿Por qué usamos OWL?*

La web semántica cuenta con un significado explícito, facilitando la tarea de las computadoras para procesar e integrar automáticamente la información que se encuentra en la Web. La web semántica se basa en la capacidad de XML para definir esquemas de etiquetas personalizados y en el enfoque flexible de RDF para representar datos.

El primer nivel por encima de RDF, requerido por la web semántica, es un lenguaje de ontologías que puede describir formalmente el significado de terminología usado en documentos Web. Si uno espera que las computadoras realicen las tareas de razonamiento sobre estos documentos, entonces el lenguaje debe ir más allá de la semántica básica de RDFS.

OWL ha sido diseñado para cumplir con tales requisitos de un lenguaje de ontologías para la Web.

2.2.3 *Orígenes*

El diseño de OWL fue el resultado de una variedad de influencias: paradigmas para la representación de conocimiento, lenguajes para ontologías existentes y lenguajes de la web semántica existentes.

OWL no fue el primer lenguaje para ontologías usadas en la Web, y su diseño fue influenciado por otros lenguajes existentes, tales como: RDFS, *SHOE* [33], *OIL [26]*, *DAML-ONT* [27] y *DAML+OIL* [35]. En particular, DAML+OIL fue una gran influencia para OWL, y la W3C[2] dicta explícitamente que el diseño de OWL debe estar basado en DAM+OIL. A su vez, DAM+OIL fue muy influenciado por el lenguaje OIL, con influencias adicionales de DAML-ONT y RDFS.

En particular, las lógicas de descripción tuvieron mucho impacto en la especificación formal de OWL, mientras que su estructura fue influenciada por el paradigma de estructuras, y la sintaxis RDF/XML estuvo dada por un requerimiento de compatibilidad con respecto a RDF (pues en la Figura 1.2 vemos que la capa de OWL está por encima de la capa de RDF). Analicemos un poco estos tres pilares fundamentales sobre los cuales OWL se ha basado.

Como mencionamos, las lógicas de descripción han tenido un fuerte impacto en el diseño de OWL; en particular sobre el formalismo de la semántica, la elección de los constructores del lenguaje y la integración de tipos de datos y valores de datos. Casos concretos como OWL_{Lite} y OWL_{DL} (ambos sublenguajes de OWL), están basados en las lógicas de descripción $\mathcal{SHIF}$ y $\mathcal{SHOIN}$, respectivamente.

En el contexto de la web semántica, donde los usuarios experimentados desearían crear o modificar ontologías, la legibilidad y la facilidad de uso son consideraciones importantes a la hora de diseñar un lenguaje de ontologías. En el diseño de OIL, estos requerimientos fueron cumplidos proveyendo una sintaxis basada en el paradigma de estructuras. Las estructuras agrupan información acerca de cada clase, haciendo que las ontologías sean más fáciles de leer y de entender. El paradigma de las estructuras ha sido utilizado en varios sistemas de representación de conocimiento, como por ejemplo la herramienta de diseño de ontologías *Protégé* [29] y OKBC.

La tercer mayor influencia en el diseño de OWL fue el requisito de mantener la mayor compatibilidad posible con los lenguajes Web existentes, en particular con RDF. Este requerimiento estaba bien fundado dado que RDF (y en particular RDFS) ya incluía varias de las características básicas requeridas para el diseño de ontologías.

A simple vista, este requerimiento podría ser satisfecho fácilmente: se le podría dar a OWL una sintaxis basada en RDF. Sin embargo, para poder proveer una máxima compatibilidad, era también necesario asegurar que la semántica de las ontologías en OWL fueran consistentes con la semántica de RDF. Esto no fue una tarea fácil de lograr, ya que OWL provee una expresividad mucho mayor que RDF. Como resultado, se implementaron tres soluciones para abordar el problema:

- OWL_{Lite}: es un subconjunto sintáctico de OWL_{DL}, adecuado cuando se busca una sintaxis más simple y una inferencia aún más manejable.

2 http://www.w3.org/2001/sw/WebOnt/charter

- OWL_{DL}: si se busca, en cambio, una sintaxis más afable o que la inferencia sea decibible, OWL_{DL} es apropiado.
- OWL_{Full}: por último, si se considera como más importante a la compatibilidad con RDF y RDFS, entonces OWL_{Full} (una extensión sintáctica y semántica de RDFS), es la solución apropiada.

Veamos a continuación algunos detalles de cada uno de estos sublenguajes de OWL.

2.2.4 *Variantes de OWL*

Encontrar el balance justo entre la expresividad del lenguaje y la eficiencia a la hora de razonar (i.e., escalabilidad) fue central para el diseño de OWL. Pues contar con construcciones complejas para representar conocimiento implícito por lo general implica un gran costo computacional o incluso indecibilidad, y por lo tanto problemas de escalabilidad.

Como mencionamos anteriormente, tres sublenguajes de OWL han sido diseñados con el fin de darle opciones al usuario en cuanto a diferentes niveles de expresividad y usabilidad:

1. **OWL_{Lite}**: Utilizado principalmente por usuarios que necesitan una clasificación jerárquica y restricciones simples. Por ejemplo, aunque da soporte a restricciones de cardinalidad, solo permite los valores 0 y 1. Dado que OWL_{Lite} está basado en la lógica de descripción $\mathcal{SHIF}$, es decidible y tiene complejidad ExpTime en cuanto a la inferencia.
2. **OWL_{DL}**: Da soporte a usuarios que requieren mayor expresividad, manteniendo completitud (está garantizado que todas las inferencias son computables) y decibilidad (todas las computaciones terminan en un tiempo finito). Es soportado por la gran mayoría de los razonadores. Su complejidad en cuanto a la inferencia es NExpTime, ya que OWL_{DL} está basado en $\mathcal{SHOIN}$.
3. **OWL_{Full}**: Diseñado para usuarios que requieren mayor expresividad y la libertad sintáctica que provee RDF, pero no da ninguna garantía computacional. No hay razonadores que puedan ser capaces de dar soporte a cada característica de OWL_{Full}. Es muy difícil para trabajar y entender semánticamente.

La versión actual de OWL soportada por la W3C es OWL 2, y su semántica está dada por la lógica de descripción $\mathcal{SROIQ}$ [47].

2.2.5 *Sintaxis de OWL*

En el Cuadro 2.2 introducimos parte de la sintaxis de OWL_{DL} y su equivalente en $\mathcal{DL}$. En el Cuadro 2.3 se presentan algunas construcciones más que pueden realizarse con OWL.

Ejemplo 2.3. A modo de ejemplo, en la Figura 2.1, vemos cómo la siguiente fórmula expresada en $\mathcal{ALC}$ $\chi = Persona \sqcap \forall tieneHijo(Doctor \sqcup \exists tieneHijo.Doctor)$, puede ser escrita en lenguaje OWL_{DL}.

SINTAXIS DE OWL_{DL}	SINTAXIS EN $\mathcal{DL}$
R someValuesFrom(C)	$\exists R.C$
R allValuesFrom(C)	$\forall R.C$
R hasValue(o)	$R : o$
complementOf(C)	$\neg C$
SubClassOf($C_1 C_2$)	$C_1 \sqsubseteq C_2$
EquivalentClasses($C_1 C_2$)	$C_1 \equiv C_2$
unionOf($C_1 C_2$)	$C_1 \sqcup C_2$
intersectionOf($C_1 C_2$)	$C_1 \sqcap C_2$
DisjointClasses($C_1 C_2$)	$C_1 \sqcap C_2 \equiv \bot$
ObjectPropery(R domain(C))	$\geq 1R \sqsubseteq C$
ObjectProperty(R range(C))	$\top \sqsubseteq \forall RC$
FunctionalProperty(R)	$\top \sqsubseteq \leq 1R$
InverseFunctionalProperty(R)	$\top \sqsubseteq \leq 1R^{-}$
maxCardinality(n R C)	$\leq nR.C$
minCardinality(n R C)	$\geq nR.C$
cardinality(n R C)	$\geq nR.C \wedge \leq nR.C$

Cuadro 2.2: Una parte de la sintaxis de OWL_{DL} y su equivalente en $\mathcal{DL}$

SINTAXIS DE OWL_{DL}	SINTAXIS EN $\mathcal{DL}$
Class(A partial $C_1...C_n$)	$A \sqsubseteq C_1 \sqcap ... \sqcap C_n$
Class(A total $C_1...C_n$)	$A \equiv C_1 \sqcap ... \sqcap C_n$
minCardinality(n R)	$\geq nR$
maxCardinality(n R)	$\leq nR$
EquivalentProperties($R_1...R_n$)	$R_1 \equiv ... \equiv R_n$
SubPropertyOf($R_1 R_2$)	$R_1 \sqsubseteq R_2$
SameIndvidual($o_1...o_n$)	$o_1 \equiv ... \equiv o_n$
DifferentIndividual($o_1...o_n$)	$o_i \neq o_j, i \neq j$
Individual(o type(C_1)...type(C_n))	$o \in C_i, i = 1, ..., n$
Individual(o value($R_1 o_1$)....value($R_n o_n$))	$(o, o_i) \in R_i$

Cuadro 2.3: Algunas construcciones extras

```
<owl:Class>
 <owl:intersectionOf rdf:parseType=" collection">
   <owl:Class rdf:about="#Persona"/>
   <owl:Restriction>
     <owl:onProperty rdf:resource="#tieneHijo"/>
     <owl:allValuesFrom>
       <owl:unionOf rdf:parseType=" collection">
         <owl:Class rdf:about="#Doctor"/>
         <owl:Restriction>
           <owl:onProperty rdf:resource="#tieneHijo"/>
           <owl:someValuesFrom rdf:resource="#Doctor"/>
         </owl:Restriction>
       </owl:unionOf>
     </owl:allValuesFrom>
   </owl:Restriction>
 </owl:intersectionOf>
</owl:Class>
```

Figura 2.1: Fórmula escrita en OWL_{DL}

3

DE LÓGICAS DE DESCRIPCIÓN A LÓGICAS MODALES

En este capítulo presentamos brevemente las lógicas modales, y luego estableceremos la conexión entre éstas y las lógicas de descripción. Esta conexión es la que luego nos permitirá transferir resultados sobre simetrías en lógicas modales a lógicas de descripción.

3.1 LÓGICAS MODALES

Comencemos presentando los conceptos principales de lógicas modales que utilizaremos en el resto de este trabajo.

Definición 3.1. *(Sintaxis). Sea* PROP $= \{p_1, p_2, ...\}$ *un conjunto contable infinito de variables proposicionales y* MOD $= \{m_1, m_2, \dots\}$ *un conjunto de símbolos de modalidades. El conjunto de las fórmulas modales básicas* FORM *se define como*

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

donde $p \in$ PROP, $m \in$ MOD, $\varphi, \psi \in$ FORM, *y* A *es el operador universal. Cuando* MOD *es un singleton, es decir, en el caso mono-modal, simplemente escribimos* $\Box$ *y* $\Diamond$*, en vez de* $[m]$*,* y $\langle m\rangle$*, respectivamente.*

Las fórmulas modales se evalúan sobre *Modelos de Kripke.*

Definición 3.2. *(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:*

- *W, el* dominio, *es un conjunto no vacío. Los elementos de W se denominan* puntos, estados, mundos, *etc.*
- *Cada* R^m *es una relación binaria sobre W.*
- *V, la* valuación, *es una función que asigna a cada elemento* $w \in W$ *un subconjunto* $V(W) \subseteq$ PROP*. Informalmente, podemos pensar a* $V(w)$ *como el conjunto de variables proposicionales que son verdaderas en w.*

Sin embargo, un modelo de Kripke puede considerarse como un modelo del tipo que fueron introducidos para las lógicas de descripción en el Capítulo 2.

Proposición 3.1. *Todo modelo de Kripke* $\mathcal{M} = \langle W, \{R^m\}_{m\in\mathsf{MOD}}, V\rangle$ *tiene asociado un modelo equivalente (en la semántica de lógicas de descripción)* $\mathcal{M}' = \langle \mathcal{I}, \Delta\rangle$*, donde* $\Delta = W$ *e* $\mathcal{I}$ *es una función de interpretación, definida como*

$$\begin{aligned} \mathcal{I}(p) &\doteq \{w \mid p \in V(w)\}, \text{ para } p \in \mathsf{PROP} \\ \mathcal{I}(R^m) &\doteq R^m \text{ para } m \in \mathsf{MOD} \end{aligned}$$

Para facilitar la transferencia de resultados de lógicas modales a lógicas de descripción, daremos la semántica de la lógica modal presentada en una notación que corresponda a la dada para lógicas de descripción.

Definición 3.3. *(Semántica). Sea $\mathcal{M} = \langle \mathcal{I}, \Delta \rangle$ un modelo, donde Δ es un conjunto no vacío e $\mathcal{I} : \mathsf{FORM} \to \mathcal{P}(\Delta)$. Sean $\varphi, \psi \in \mathsf{FORM}$ y $m \in \mathsf{MOD}$. Definimos $\mathcal{I}$ recursivamente de la siguiente manera*

$$
\begin{aligned}
\mathcal{I}(\neg\varphi) &\doteq \Delta \setminus \mathcal{I}(\varphi) \\
\mathcal{I}(\varphi \vee \psi) &\doteq \mathcal{I}(\varphi) \cup \mathcal{I}(\psi) \\
\mathcal{I}(\varphi \wedge \psi) &\doteq \mathcal{I}(\varphi) \cap \mathcal{I}(\psi) \\
\mathcal{I}([m]\varphi) &\doteq \{x \in \Delta \mid \forall y \in \Delta, (x,y) \in \mathcal{I}(R^m) \implies y \in \mathcal{I}(\varphi)\} \\
\mathcal{I}(\langle m \rangle \varphi) &\doteq \{x \in \Delta \mid \exists y \in \Delta, (x,y) \in \mathcal{I}(R^m) \wedge y \in \mathcal{I}(\varphi)\} \\
\mathcal{I}(A(\varphi)) &\doteq \begin{cases} \Delta & si\ \mathcal{I}(\varphi) = \Delta \\ \emptyset & si\ \mathcal{I}(\varphi) \neq \Delta \end{cases}
\end{aligned}
$$

3.2 HACIA LÓGICAS DE DESCRIPCIÓN

La Definición 3.3 nos permite aplicar modelos de lógicas de descripción a fórmulas de lógicas modales. A continuación, veremos cómo podemos reescribir fórmulas modales cómo fórmulas de lógicas de descripción.

3.2.1 *El puente de Schild*

El lenguaje para definir conceptos en lógicas de descripción es muy similar al de las lógicas modales. Esta similitud fue observada por Schild en [55], quien hizo uso de esto para transferir resultados de complejidad y axiomatización de lógicas modales a lógicas de descripción.

Schild también notó que el nexo entre lógicas modales y lógicas de descripción puede establecerse solo al nivel de *satisfacibilidad de concepto*s. La lógica modal básica no es tan expresiva como el razonamiento sobre ABox o inferencia sobre definiciones de una TBox.

3.2.1.1 *Lógicas modales en términos de lógicas de descripción*

Ahora trazaremos el nexo sintáctico entre las lógicas modales y las lógicas de descripción. Como mencionamos anteriormente, distintas clases de lenguajes en lógicas de descripción cuentan con diferente expresividad, y por lo tanto, complejidad computacional. Por ahora nos concentraremos solo en $\mathcal{ALC}$.

Las expresiones entre conceptos en $\mathcal{ALC}$ podrían pensarse como fórmulas modales, como muestra el Cuadro 3.1.

$\mathcal{ML}$	$\mathcal{ALC}$
p_i	C_i, y C_i es un concepto atómico
$\neg\varphi$	$\neg C$, y C es un concepto
$\varphi \vee \psi$	$C_j \sqcup C_i$, con Cj, C_i conceptos
$\varphi \wedge \psi$	$C_j \sqcap C_i$, con Cj, C_i conceptos
$\langle R \rangle \varphi$	$\exists R.C_j$, y R es un rol atómico y C_j es un concepto
$[R]\varphi$	$\forall R.C_j$, y R es un rol atómico y C_j es un concepto

Cuadro 3.1: Relación entre $\mathcal{ML}$ y $\mathcal{ALC}$

Es importante notar que el Cuadro 3.1 muestra la conexión entre el lenguaje de concepto de $\mathcal{ALC}$ y el lenguaje modal básico (i.e., sin el operador A), presentada originalmente en [55]. Para obtener una conexión que también incluya la TBox, hace falta el operador A, lo que discutiremos en la siguiente sección.

Para representar la ABox es necesario contar con los operadores modales que tienen las lógicas híbridas [2]. Sin embargo esta última conexión queda fuera de lo investigado en esta tesis.

3.2.2 *De OWL a lógicas modales*

En la sección anterior presentamos la conexión existente entre $\mathcal{ALC}$ y la lógica modal básica. También mencionamos que no era posible incluir a la TBox ya que necesitábamos el operador universal A. Completemos ahora la conexión, pero en vez de hacerlo sobre $\mathcal{ALC}$, lo haremos partiendo de OWL; ya que nos interesa buscar simetrías en ontologías reales que estén implementadas y sean utilizadas por los razonadores de lógicas de descripción, y las mismas están escritas con el lenguaje OWL (OWL_{DL}, para ser precisos).

Definición 3.4. *(Función de traducción Ψ). Sea $\mathcal{F}_{\mathcal{DL}}$ el conjunto de fórmulas de lógicas de descripción expresadas en OWL,* FORM *el conjunto de fórmulas modales y sea $D_\Psi \subseteq \mathcal{F}_{\mathcal{DL}}$ el dominio de la función Ψ. Sea $\mathcal{R}^m$ una propiedad, $\mathcal{C}$ un concepto atómico y $\mathcal{C}_i, \mathcal{C}_j$ conceptos para $i, j \in \mathbb{N}$. Entonces $\Psi : D_\Psi \to$ FORM queda definida de la siguiente manera:*

$$
\begin{aligned}
\Psi(\mathcal{R}\, someValuesFrom\, \mathcal{C}) &\doteq \langle \mathcal{R} \rangle \Psi(\mathcal{C}) \\
\Psi(\mathcal{R}\, allValuesFrom\, \mathcal{C}) &\doteq [\mathcal{R}] \Psi(\mathcal{C}) \\
\Psi(\mathcal{R}\, hasValue\, \mathcal{C}) &\doteq [\mathcal{R}] \Psi(\mathcal{C}) \wedge \langle \mathcal{R} \rangle \Psi(\mathcal{C}) \\
\Psi(complementOf(\mathcal{C})) &\doteq \neg \Psi(\mathcal{C}) \\
\Psi(SubClassOf(\mathcal{C}_1\, \mathcal{C}_2)) &\doteq A(\Psi(\mathcal{C}_1) \implies \Psi(\mathcal{C}_2)) \\
\Psi(EquivalentClasses(\mathcal{C}_1 \equiv \mathcal{C}_2)) &\doteq \begin{array}{l} \Psi(SubClassOf(\mathcal{C}_1\, \mathcal{C}_2)) \wedge \\ \Psi(SubClassOf(\mathcal{C}_2\, \mathcal{C}_1)) \end{array} \\
\Psi(unionOf(\mathcal{C}_1\, \mathcal{C}_2)) &\doteq \Psi(\mathcal{C}_1) \vee \Psi(\mathcal{C}_2) \\
\Psi(intersectionOf(\mathcal{C}_1\, \mathcal{C}_2)) &\doteq \Psi(\mathcal{C}_1) \wedge \Psi(\mathcal{C}_2)
\end{aligned}
$$

$$
\begin{aligned}
\Psi(\mathit{DisjointClasses}(\mathcal{C}_1\,\mathcal{C}_2)) &\doteq A(\Psi(\mathcal{C}_1) \Leftrightarrow \neg\Psi(\mathcal{C}_2)) \\
\Psi(\mathit{ObjectProperty}(\mathcal{R}\,\mathit{domain}\,(\mathcal{C}))) &\doteq A(\langle\mathcal{R}\rangle\top \implies \Psi(\mathcal{C})) \\
\Psi(\mathit{ObjectProperty}(\mathcal{R}\,\mathit{range}\,(\mathcal{C}))) &\doteq A(\langle\mathcal{R}^-\rangle\top \implies \Psi(\mathcal{C})) \\
\Psi(\mathit{FunctionalProperty}(\mathcal{R})) &\doteq A(\langle\mathcal{R}\rangle\top \implies [\mathcal{R}]\top) \\
\Psi(\mathit{InverseFunctionalProperty}(\mathcal{R})) &\doteq A(\langle\mathcal{R}^-\rangle\top \implies [\mathcal{R}^-]\top) \\
\Psi(\mathit{maxCardinality}\ n\,R\,\mathcal{C}) &\doteq \langle MAX\,n\,R\rangle\Psi(\mathcal{C}) \\
\Psi(\mathit{minCardinality}\ n\,R\,\mathcal{C}) &\doteq \langle MIN\,n\,R\rangle\Psi(\mathcal{C}) \\
\Psi(\mathit{cardinality}\ n\,R\,\mathcal{C}) &\doteq \langle =\ n\,R\rangle\Psi(\mathcal{C})
\end{aligned}
$$

La función de traducción Ψ tiene la siguiente propiedad:

Teorema 3.1. *Sea $\mathcal{M} = \langle\Delta, \mathcal{I}\rangle$ un modelo y $\varphi \in \mathcal{F}_{\mathcal{DL}}$, entonces vale que $\mathcal{I}(\varphi) = \mathcal{I}(\Psi(\varphi))$.*

3.2.3 *Análisis de la función de traducción*

Si bien no probaremos formalmente el Teorema 3.1, se puede deducir la correctitud de la traducción de forma intuitiva analizando algunas de las diferentes reglas que la componen. Para ello utilizaremos también el Cuadro 2.2, que relaciona OWL con fórmulas de $\mathcal{DL}$.

1. $\mathbf{\Psi}(\mathbf{complementOf}\,(\mathcal{C})) \doteq \neg\mathbf{\Psi}(\mathcal{C})$: El complemento de conjuntos, ¬C en $\mathcal{DL}$, se traduce como la negación de una fórmula de $\mathcal{ML}$.

2. $\mathbf{\Psi}(\mathbf{unionOf}(\mathcal{C_1}\mathcal{C_2})) \doteq \mathbf{\Psi}(\mathcal{C_1}) \vee \mathbf{\Psi}(\mathcal{C_2})$: $\mathcal{C}_1 \sqcup \mathcal{C}_2$ en sintaxis de $\mathcal{DL}$. Debemos recordar que a nivel de modelos siempre estamos trabajando con conjuntos y propiedades sobre éstos, por lo tanto en $\mathcal{DL}$ las propiedades que puedan cumplirse sobre la unión de dos conjuntos, será lo mismo que en $\mathcal{ML}$ se cumpla sobre algunos de los dos.

3. $\mathbf{\Psi}(\mathcal{R}\,\mathbf{someValuesFrom}\,\mathcal{C}) \doteq \langle\mathcal{R}\rangle\mathbf{\Psi}(\mathcal{C})$: En $\mathcal{DL}$ expresada como $\exists R.C$, esta expresión se interpreta como "los elementos *a* tales que existe un elemento *b* y el par *(a, b)* forma parte de *R* y *b* está en el conjunto *C*". En $\mathcal{ML}$ esto está representado por su cuantificador existencial $\langle R\rangle$.

4. $\Psi(\mathcal{R}\,\mathbf{allValuesFrom}\,\mathcal{C}) \doteq [\mathcal{R}]\Psi(\mathcal{C})$: De forma análoga, aquí se expresa "los elementos *a* tales que para todo elemento *b* tal que *(a, b)* esté en *R* entonces se da que *b* pertence a *C*", lo que se corresponde con el cuantificador universal de $\mathcal{ML}$ dado por el operador $[R]$.

5. $\mathbf{\Psi}(\mathbf{SubClassOf}(\mathcal{C_1}\mathcal{C_2})) \doteq \mathbf{A}(\mathbf{\Psi}(\mathcal{C_1}) \implies \mathbf{\Psi}(\mathcal{C_2}))$: $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2$ si lo quisiéramos expresar en $\mathcal{DL}$. Se debe verificar que en todo el modelo siempre que un elemento pertenece a C_1, también pertenece a C_2. Para asegurar que la inclusión se verifica en todo el modelo, es necesario utilizar el operador universal A.

6. $\mathbf{\Psi}(\mathbf{EquivalentClasses}(\mathcal{C_1}\mathcal{C_2})) \doteq \mathbf{\Psi}(\mathbf{SubClassOf}(\mathcal{C_1}\,\mathcal{C_2}))\wedge$
$\mathbf{\Psi}(\mathbf{SubClassOf}(\mathcal{C_2}\,\mathcal{C_1}))$: La equivalencia de conjuntos no es nada menos que una doble inclusión.

7. $\mathbf{\Psi}(\mathbf{DisjointClasses}(\mathcal{C}_1\,\mathcal{C}_2)) \doteq \mathbf{A}(\mathbf{\Psi}(\mathcal{C}_1) \Leftrightarrow \neg\mathbf{\Psi}(\mathcal{C}_2))$: En el caso de conjuntos disjuntos, solo una de las dos vale.

8. $\mathbf{\Psi}(\mathbf{minCardinality}\ n\,R\,\mathcal{C}) \doteq \langle MIN\,n\,R\,\rangle\Psi(\mathcal{C})$: consideraremos a los casos de cardinalidad como una relación más, distinguiendo cada expresión de cardinalidad por separado.

4

SIMETRÍAS EN LÓGICAS DE DESCRIPCIÓN

En este capítulo hablaremos de simetrías en lógicas de descripción. Primero daremos una noción de simetrías en el contexto de lógicas modales, luego utilizando el nexo entre lógicas de descripción y lógicas modales dado en el capítulo anterior mostraremos que los resultados sobre simetrías en lógicas modales se pueden transferir a lógicas de descripción. Esto nos permitirá realizar la detección de simetrías en fórmulas de lógicas de descripción usando las técnicas disponibles en lógicas modales.

4.1 SIMETRÍAS EN LÓGICAS MODALES

Las simetrías en lógicas modales fueron extensamente estudiadas en [45]. A continuación introduciremos los conceptos principales.

Definición 4.1. *(Permutación). Una* permutación *de un conjunto A es una biyección $\sigma : A \to A$.*

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

Definición 4.2. *(Notación cíclica). Sea A un conjunto finito y sean $a_1, ..., a_n$ elementos distintos de A. La expresión $(a_1\ a_2\ ...\ a_n)$ es un* ciclo *y denota la acción de mapear $a_1 \to a_2, a_2 \to a_3, ..., a_{n-1} \to a_n, a_n \to a_1$. 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.

Definición 4.3. *(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.*

Definición 4.4. *(Permutación de literales proposicionales). Una* permutación de literales proposicionales *es una función biyectiva σ:* $\mathsf{PLIT} \longmapsto \mathsf{PLIT}$. *Dado un conjunto de literales proposicionales L, $\sigma(L) = \{\sigma(l) \mid l \in L\}$.*

La definición que acabamos de dar 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 en la cual estamos trabajando.

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

Definición 4.5. *(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* $S \cup \{\neg p \mid p \in \mathsf{PROP} \setminus S\}$.

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

Definición 4.6. *(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 [28, 57], 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 4.7. *(Permutación de una fórmula). Sea* φ *una fórmula en CNF modal y* σ *una permutación. Definimos* $\sigma(\varphi)$ *recursivamente como*

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

Definición 4.8. *(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 las disyunciones en* φ *son representadas como conjuntos.*

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

Definición 4.9. *(Permutación de un modelo). Sea* σ *una permutación y* $\mathcal{M} = \langle \Delta, \mathcal{I} \rangle$ *un modelo. Luego* $\sigma(\mathcal{M}) = \langle \Delta, \mathcal{I}' \rangle$, *donde,* $\mathcal{I}'(p) = \mathcal{I}(\sigma^{-1}(p))$ *para todo* $p \in \mathsf{PROP}$.

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

Proposición 4.1. *Sea* σ *una permutación consistente,* φ *una fórmula en CNF modal y* $\mathcal{M} = \langle \Delta, \mathcal{I} \rangle$ *un modelo. Luego* $\mathcal{M} \models \varphi$ *si y solo si* $\sigma(\mathcal{M}) \models \sigma(\varphi)$.

Demostración: Ver [45].

Definición 4.10. *(Conjunto de Modelos de φ). Sea φ una fórmula y $\mathcal{M}$ un modelo. Denotaremos con $Mods(\varphi)$ al conjunto $\{\mathcal{M} \mid \mathcal{M} \models \varphi\}$.*

Corolario 4.1. *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 4.1 nos indica que, en la lógica modal básica, el grupo de simetrías de una fórmula φ actuando sobre el conjunto de modelos lo particiona de forma tal que las clases de equivalencia 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 4.1. *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 [45].

El Teorema 4.1 nos provee un 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 [9] o un paso de resolución [3]. 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 [6].

El Corolario 4.1 junto con el Teorema 4.1 son los resultados claves que permiten utilizar las simetrías de una fórmula en la lógica modal básica.

Gracias a la función de traducción de la Definición 3.4 estos Teoremas se pueden transferir a las lógicas de descripción. Es decir, podemos obtener una contraparte del Corolario 4.1 y del Teorema 4.1.

Teorema 4.2. *Sea $\mathcal{KB}$ una base de conocimiento, φ una fórmula como en la Definición 2.2 , Ψ la función de traducción de la Definición 3.4 y σ una simetría de $\Psi(\mathcal{KB})$, luego se cumple que*

$$\mathcal{KB} \models \varphi \iff \mathcal{KB} \models \sigma(\varphi).$$

4.2 DETECTANDO SIMETRÍAS

El primer paso para utilizar las simetrías en cualquier lógica es detectarlas. En esta sección nos enfocaremos en detectar simetrías en ontologías. Para ello veremos cómo reutilizar técnicas de detección basadas en grafos utilizadas para detectar simetrías en fórmulas modales. Nuevamente la traducción del Capítulo 3 nos permitirá hacer la transferencia de los resultados obtenidos en [45] para lógicas modales a lógicas de descripción.

La detección de simetrías basada en grafos es la técnica más común para detectar simetrías en fórmulas y ha encontrado aplicación en diferentes dominios [1, 20, 38].

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. Y segundo, la disponibilidad de herramientas de detección de automorfismos de grafos muy eficientes [37, 53].

4.2.1 *Detección de simetrías en lógicas modales*

A continuación presentaremos uno de los algoritmos de detección de simetrías en lógicas modales introducidos en [45].

Definición 4.11. *(Profundidad Modal). La* profundidad modal *de una fórmula φ (notación $md(\varphi)$) es una función que va de fórmulas a los números naturales definida como sigue:*

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

A menos que se indique lo contrario, trabajaremos con fórmulas en CNF modal, 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 4.12. *(Variables proposicionales de una fórmula). Sea φ una fórmula en CNF modal. Con $Prop(\varphi)$ denotamos al conjunto de variables proposicionales que ocurren en φ independientemente de la profundidad modal a la cual ocurren.*

Definición 4.13. *(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 cómo colorear los nodos. Para ello definamos una *función de tipado:*

Definición 4.14. *(Función de tipado). Sea s: $\mathsf{MOD} \times \{0,1\} \to \mathbb{N} \setminus \{0,1\}$ una función inyectiva arbitraria y sea $t : Sub(\varphi) \to \mathbb{N}$ una función parcial definida como*

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

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

Definición 4.15. *(Algoritmo de construcción de grafos). Sea φ una fórmula en CNF modal y t una función de tipado. El grafo $G(\varphi) = (V, E)$ se construye de la siguiente forma:*

1. *Para cada variable proposicional $p \in Prop(\varphi)$:*
   a) *Agregar dos nodos de color (tipo) 0: uno para el literal positivo p y otro para el literal negativo $\neg p$.*
   b) *Agregar un arco E entre el literal positivo y el literal negativo para asegurar consistencia Booleana.*
2. *Para cada cláusula top C en φ :*
   a) *Agregar un nodo* cláusula *de color t(C).*
   b) *Para cada literal proposicional l que ocurre en C, agregar un arco E 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 entre el nodo para C y el nodo para C'.*
      3) *Repetir el proceso desde el punto 2b por cada literal (proposicional o modal) que ocurre en C'.*

Para una fórmula con P variables proposicionles, C cláusulas top, M cláusulas modales, y R modalidades, esta construcción produce un grafo con $2 + 2R$ colores y $(2P + C + M)$ nodos.

Ejemplo 4.1. *Consideremos la fórmula modal $\varphi = (a \vee [m](b \vee \neg[m]c)) \wedge (b \vee [m](a \vee \neg[m]c))$.*

La Figura 4.1 muestra su grafo $G(\varphi)$ asociado construido usando el algoritmo de la Definición 4.15 (los colores están representados por las diferentes formas en la figura).

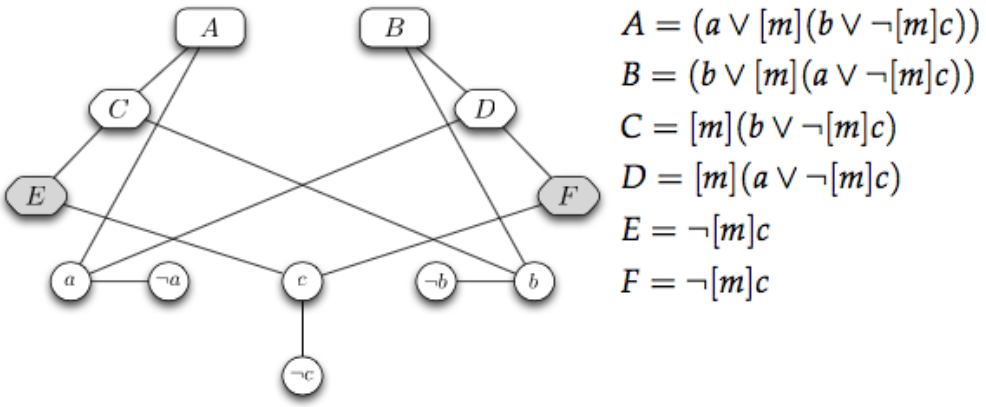

Figura 4.1: Grafo obtenido usando el algoritmo de construcción de grafos

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.

En [45] se prueba que el algoritmo de la Definición 4.15, es correcto.

Teorema 4.3. *Sea φ una fórmula en CNF modal y $G(\varphi) = (V, E)$ el grafo coloreado construido a partir de la Definición 4.15. Luego, toda simetría σ de φ se corresponde uno-a-uno con un automorfismo π de $G(\varphi)$.*

Demostración: ver [45].

El Teorema 4.3 nos asegura que el algoritmo es correcto, es decir, que no se detectarán simetrías espúreas (o sea, automorfismos del grafo que no son simetrías de la fórmula).

4.2.2 *Detección de simetrías en lógicas de descripción*

Ahora bien, ¿cómo podemos detectar simetrías en lógicas de descripción? Gracias a la función de traducción de la Definición 3.4 podemos utilizar el algoritmo de la Definición 4.15 para hacer la detección de simetrías en ontologías.

Para ello, dada una ontología primero debemos traducirla a una fórmula modal usando la función de traducción. Luego podemos aplicar la construcción descrita anteriormente para detectar simetrías en dicha fórmula modal y finalmente debemos traducir esas simetrías a simetrías de nuestra ontología.

Por lo tanto, si ϑ es alguna ontología de interés y Ψ nuestra función de traducción definida en el Capítulo 3, la fórmula modal Γ sobre la que buscaremos simetrías estará dada por

$$\Gamma = \bigwedge_{\alpha \in TBox(\vartheta)} \Psi(\alpha).$$

UNIENDO LAS PIEZAS

Este capítulo hizo foco en la detección de simetrías en lógicas modales, y en la transferencia de resultados teóricos de lógicas modales a lógicas de descripción. Estos resultados permiten el uso de algoritmos de detección de simetrías en lógicas modales en lógicas de descripción.

Primero dimos una definición formal de lo que es una simetría. Luego, utilizamos el algoritmo de la Definición 4.15 para construir un grafo a partir de una fórmula en CNF modal. Gracias al Teorema 4.3, vemos que hay una equivalencia entre buscar simetrías en la fórmula, a buscar automorfismos en el grafo generado por dicha fórmula.

Por último, para la detección de simetrías en el contexto de las lógicas de descripción, nos valimos de la función Ψ definida en el Capítulo 3, para pasar fórmulas de lógicas de descripción a fórmulas modales, y luego aplicar los procedimientos para la detección de simetrías ya descritos.

5 DETALLES DE IMPLEMENTACIÓN

Ya contamos con todas las bases teóricas para poder buscar simetrías en ontologías expresadas en el lenguaje OWL. En este capítulo presentamos el conjunto de herramientas utilizadas para dicha tarea.

5.1 ARQUITECTURA GENERAL

El siguiente diagrama describe los principales componentes así como el flujo de la herramienta de detección desarrollada, disponible en [52].

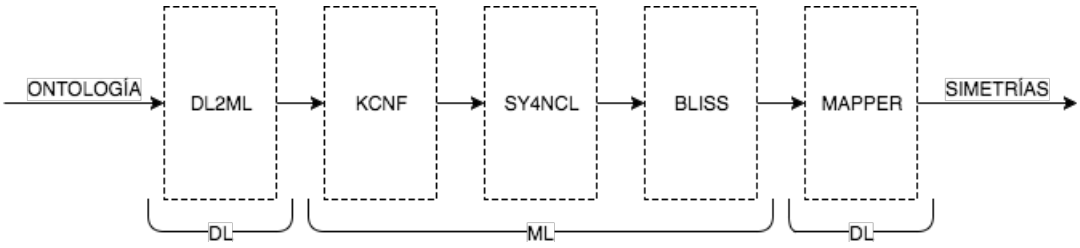

Figura 5.1: Arquitectura general para la detección de simetrías

Como dato de entrada, nuestra herramienta recibe una ontología en formato OWL. El módulo ***DL2ML***, implementado en *Scala* [54], se encarga de manipular esta ontología, cargándola en memoria y realizando una serie de operaciones que describiremos a continuación. Para ello se utiliza *Scowl* [56] una implementación de la *OWL API [60]*.

La funcionalidad de DL2ML puede dividirse en dos tareas: primero toma la ontología de entrada y extrae todos los axiomas de la TBox. Luego, a cada axioma de la TBox se lo examina recursivamente utilizando la función de traducción de la Definición 3.4 para generar una fórmula modal en formato *intohylo* [34].

El módulo ***KCNF*** toma la fórmula modal generada por DL2ML y devuelve una fórmula, semánticamente igual, en CNF modal.

Una vez que contamos con nuestra fórmula en CNF modal, el módulo ***SY4NCL***, desarrollado en Haskell [31], construye el grafo de detección de simetrías, siguiendo el algoritmo descrito en la Definición 4.15 y lo disponibiliza en formato Bliss [37] junto con un mapeo entre nodos y literales y estadísticas sobre el grafo.

El módulo ***BLISS*** toma la especificación de un grafo y realiza la búsqueda de automorfismos en éste. Un detalle importante es que BLISS solo devuelve los automorfismos *generadores*. Éstos son aquellos automorfismos que permiten generar todos los automorfismos del grupo.

Si el grafo tiene automorfismos no triviales, entonces el módulo ***MAPPER*** reconstruye las simetrías de la ontología primero utilizando el mapeo entre nodos y literales generado por SY4NCL, y luego realizando un mapeo *de vuelta* entre símbolos proposicionales a conceptos dentro de nuestra ontología.

Cabe mencionar que el componente DL2ML fue desarrollado específicamente para el trabajo de esta tesis, mientras que los módulos KCNF y SY4NCL fueron adaptados para que funcionen en el caso de lógicas multi modales. La herramienta BLISS no requirió de ninguna modificación.

Por último, es importante resaltar que las simetrías pueden ser detectadas de manera independiente a las tareas de inferencia, i.e., se pueden detectar las simetrías previamente y utilizarlas luego para cada consulta de inferencia. Esto es de gran importancia ya que no resultaría muy práctico tener que realizar la detección de simetrías cada vez que se le hace una consulta a la base de conocimiento, dado que obtener las simetrías puede ser una tarea relativamente costosa.

5.2 PROBANDO LAS HERRAMIENTAS

Para ilustrar el funcionamiento de nuestra herramienta veamos cómo es una ejecución de principio a fin de la misma.

Ejemplo 5.1. Consideremos la siguiente base de conocimiento $\mathcal{KB} = \langle TBox, ABox \rangle$ donde la ABox es vacía y la TBox está dada por las definiciones de la Figura 5.2.

$$
\begin{aligned}
gato &\sqsubseteq mamífero \\
perro &\sqsubseteq mamífero \\
caballo &\sqsubseteq mamífero \\
loro &\sqsubseteq ave \\
\neg(mamífero &\equiv ave) \\
pichón &\equiv loro \sqcap \exists críaDe.loro \\
gatito &\equiv gato \sqcap \exists críaDe.gato \\
cachorro &\equiv perro \sqcap \exists críaDe.perro \\
potrillo &\equiv caballo \sqcap \exists críaDe.caballo \\
gato &\sqsubseteq mamífero \sqcap \exists cuadrúpedo.mamífero \\
perro &\sqsubseteq mamífero \sqcap \exists cuadrúpedo.mamífero \\
caballo &\sqsubseteq mamífero \sqcap \exists cuadrúpedo.mamífero
\end{aligned}
$$

Figura 5.2: Descripción de la TBox

Notar que las clases *caballo*, *perro* y *gato* comparten la mismas propiedades, o sea, todas éstas son subclases de los *mamíferos* y a la vez son *cuadrúpedos*.

Ahora representamos la ontología de la Figura 5.2 en lenguaje OWL.

A modo ilustrativo, primero mostraremos cómo se puede definir una ontología en OWL programáticamente.

La Figura 5.3 muestra el código en Scala que construye la ontología utilizando Scowl.

```
1. // Creación de la ontología
2. val manager = OWLManager.createOWLOntologyManager()
3. val ontology = manager.createOntology()
4. val prfx = "http://www.famaf.unc.edu.ar/giovannirescia/test.owl"
5.
6. // Definición de clases
7. val perro = Class(prfx#perro)
8. val cachorro = Class(prfx#cachorro)
9. val gato = Class(prfx#gato)
10. val gatito = Class(prfx#gatito)
11. val caballo = Class(prfx#caballo)
12. val potrillo = Class(prfx#potrillo)
13. val loro = Class(prfx#loro)
14. val pichón = Class(prfx#pichón)
15. val ave = Class(prfx#ave)
16. val mamífero = Class(prfx#mamífero)
17.
18. // Definición de Propiedades
19. val cuadrúpedo = ObjectProperty(prfx#cuadrúpedo)
20. val críaDe = ObjectProperty(prfx#críaDe)
21.
22. var xs = ListBuffer[OWLAxiom]
23.
24. // Relaciones entre las clases
25. xs += gato SubClassOf mamífero
26. xs += perro SubClassOf mamífero
27. xs += caballo SubClassOf mamífero
28. xs += loro SubClassOf ave
29. xs += ave DisjointWith mamífero
30. xs += pichón EquivalentTo (loro and (críaDe some loro))
31. xs += gatito EquivalentTo (gato and (críaDe some gato))
32. xs += cachorro EquivalentTo (perro and (críaDe some perro))
33. xs += potrillo EquivalentTo (caballo and (críaDe some caballo))
34. xs += gato SubClassOf (mamífero and (cuadrúpedo some mamífero))
35. xs += perro SubClassOf (mamífero and (cuadrúpedo some mamífero))
36. xs += caballo SubClassOf (mamífero and (cuadrúpedo some mamífero))

37.
38. // Guardamos los cambios en la ontología
39. xs.foreach(axiom => manager.addAxiom(ontology, axiom))
40. manager.saveOntology(ontology, file)
```

Figura 5.3: Definición de la ontología en Scala

Hasta ahora lo que hicimos fue:

- En la línea 2, crear un *administrador de ontologías*, que nos permitirá crear y manipular ontologías.
- En la línea 3, creamos una ontología.

- Entre las líneas 6 y 16, creamos las clases de interés como *perro, gato, caballo* y *loro*, cada una con su respectiva cría. También las clases a las que esos animales pertenecen, o sea, *mamífero* y *ave*.
- Las líneas 19 y 20, definen propiedades que aplicaremos a las clases definidas.
- Entre las líneas 25 y 36 definimos los axiomas de interés: cómo se relacionan las clases entre sí y qué propiedad afecta a cuál clase.
- En la línea 39 utilizamos el administrador de ontologías para cargar todos los axiomas definidos.
- En la línea 40 guardamos la ontología en disco.
- Resumamos los datos más relevantes obtenidos con respecto a la ontología:
  - *Expresividad:* $\mathcal{AL}$
  - *Cantidad de axiomas en nuestra ontolgía:* 12
  - *Tipos de axiomas: SubClassOf, EquivalentClasses, DisjointClasses*

Nota: la creación de esta ontología es a modo ilustrativo. Para fines prácticos las ontologías están ya especificadas en OWL y las mismas son parseadas por el módulo DL2ML.

Parte del código XML que se generó para nuestra ontología está descrito en la Figura 5.4.

Ahora que tenemos nuestra ontología en formato OWL, corramos nuestra herramienta y veamos qué obtenemos.

El módulo DL2ML toma la ontología de la Figura 5.4 y genera la fórmula modal de la Figura 5.5.

```
begin (A(P8 --> P11));
((A(P5 --> (P8 ^ (<R1>P8)))) ^ (A((P8 ^ (<R1>P8)) --> P5)));
((A(P8 --> (P11 ^ (<R2>P11)))) ^ (A((P11 ^ (<R2>P11)) --> P8)));
...
((A(P3 --> (P11 ^ (<R2>P11)))) ^ (A((P11 ^ (<R2>P11)) --> P3)))
end
```

Figura 5.5: Fórmula modal generada por DL2ML

El módulo KCNF toma la fórmula modal de la figura Figura 5.5 y la transforma en la fórmula modal de la Figura 5.6.

```
begin
A (P3 v -P17);
A (P7 v P11);
...
A (-P18 v -[R1]-P10);
A (-P19 v -[R2]-P11)
end
```

Figura 5.6: Versión CNF modal de la fórmula generada por KCNF

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.w3.org/2002/07/owl#"
        xml:base="http://www.w3.org/2002/07/owl"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns:xml="http://www.w3.org/XML/1998/namespace"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
        <Ontology/>

    <!-- http://www.famaf.unc.edu.ar/giovanni/test.owl#cuadrúpedo -->
        <ObjectProperty rdf:about="http://www.famaf.unc.edu.ar/giovanni/test.
            owl#cuadrúpedo"/>

        <!-- http://www.famaf.unc.edu.ar/giovanni/test.owl#newBorn -->
        <ObjectProperty rdf:about="http://www.famaf.unc.edu.ar/giovanni/test.
            owl#críaDe"/>

    <!-- http://www.famaf.unc.edu.ar/giovanni/test.owl#aves -->
    <Class rdf:about="http://www.famaf.unc.edu.ar/giovanni/test.owl#aves">
                <disjointWith rdf:resource="http://www.famaf.unc.edu.ar/
                    giovanni/test.owl#mamíferos"/>
        </Class>

...

        <!-- http://www.famaf.unc.edu.ar/giovanni/test.owl#pichón -->

        <Class rdf:about="http://www.famaf.unc.edu.ar/giovanni/test.owl#pichón
            ">
        <equivalentClass>
            <Class>
                <intersectionOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="http://www.famaf.unc.edu.ar/
                        giovanni/test.owl#loro"/>
                        <Restriction>
                                <onProperty rdf:resource="http://www.famaf.unc
                                    .edu.ar/giovanni/test.owl#críaDe"/>
                                  <someValuesFrom rdf:resource="http://www.
                                      famaf.unc.edu.ar/giovanni/test.owl#loro
                                      "/>
                            </Restriction>
                </intersectionOf>
                        </Class>
                </equivalentClass>
        </Class>
</rdf:RDF>
<!-- Generated by the OWL API (version 4.2.1.20160306-0044) https://github.com
    /owlcs/owlapi -->
```

Figura 5.4: Código XML generado para la ontología definida

El módulo SY4NCL toma la fórmula CNF modal de la Figura 5.6 y genera el grafo de la Figura 5.7 en formato bliss, junto con el mapeo de literales a nodos de la Figura 5.8.

```
p edge 128 156
n 1 1
n 2 3
...
e 1 2
e 3 4
e 2 3
...
e 128 10
```

Figura 5.7: Grafo (en formato BLISS) generado por SY4NCL

```
<md> <lit> <node_id>
0 19 46
0 18 28
0 17 6
...
0 -3 4
0 -4 61
...
0 -18 27
0 -19 45
```

Figura 5.8: Mapeo de nodos a literales generado por SY4NCL

La especificación del grafo de la Figura 5.8 es tomada por el módulo BLISS, el cual realiza la detección de automorfismos en el grafo y genera como salida estadísticas junto con la información de los generadores (si los hubiese) tal como se presenta en la Figura 5.9.

```
Computation time: 0.00705 sec
Color count:[38,36,38,1,1,0,0,3,3,0,0,4,4,0,0]
|Nodes|: 128
|Edges|: 156
Generator: (1,51)(2,52)(3,22)(4,21)(5,53)(6,54)(13,19)(14,20)(23,35)(24,36)
    (25,37)(26,38)(59,65)(60,66)(61,67)(62,68)(63,77)(64,78)(79,102)(80,103)
    (81,104)(84,93)(85,94)(86,95)(108,114)(109,115)(110,116)(111,126)(112,127)
    (113,128)
...
Generators:     1
Max level:      1
|Aut|:          2
Total time:     0.00 seconds
```

Figura 5.9: Output de BLISS

Observando las estadísticas generadas por BLISS podemos conocer que el grafo explorado tiene 123 nodos (*nodes*) y 156 aristas (*edges*), que el grafo cuenta con un automorfismo generador (*generators*), que el tamaño del grupo de automorfismos, $|A|$, es 2 y que el tiempo que llevó computar el generador fue de 0.00705 segundos.

Como vemos, BLISS fue capaz de encontrar autoformismos en el grafo. El siguiente paso es traducir los automorfismos a simetrías de nuestra $\mathcal{KB}$.

El módulo MAPPER toma las simetrías encontradas por BLISS y realiza el mapeo a simetrías de la ontología de entrada. La Figura 5.10 muestra la simetría resultante.

```
(gato perro)(caballo gato)(gatito cachorro)(potrillo gatito)
```

Figura 5.10: Simetría entre los conceptos

Como era de esperarse, encontramos que *gato*, *perro* y *caballo* son simétricos, en el sentido que cumplen con las mismas propiedades. Lo mismo pasa con *gatito*, *cachorro* y *potrillo*.

Ahora bien, ¿qué pasa si agregamos una propiedad nueva a nuestra $\mathcal{KB}$? Por ejemplo, agreguemos el axioma de la Figura 5.11

```
val nueveVidas = ObjectProperty(prfx#tiene9vidas)

xs += gato EquivalentTo (mamífero and (nueveVidas some  mamífero))
```

Figura 5.11: Axioma para romper algunas simetrías

Si volvemos a ejecutar las herramientas encontramos la simetría de la Figura 5.12.

```
(perro caballo)(cachorro potrillo)
```

Figura 5.12: Nueva simetría obtenida

Como era de esperar ahora nuestra herramienta ya no encuentra como simétricas a la clase de *gato* con *perro* o *caballo*, porque ésta tiene una propiedad que las otras clases no tienen. También es importante recalcar que la clase *gatito* dejó de ser equivalente a las clases *potrillo* y *cachorro*, esto pasa porque, aunque no le hayamos agregado ninguna propiedad explícitamente, esta clase comparte propiedades de la clase *gato* y por lo tanto comparte la propiedad *nueveVidas*.

6 ANÁLISIS DE SIMETRÍAS EN ONTOLOGÍAS EXISTENTES

En este capítulo presentamos los resultados experimentales obtenidos luego de ejecutar la herramienta presentada en el capítulo anterior, sobre ontologías existentes y en uso, obtenidas de la Web[1].

6.1 DESCRIPCIÓN DE LAS ONTOLOGÍAS UTILIZADAS

Comenzaremos por presentar el conjunto de ontologías utilizadas para evaluar nuestra herramienta. Todas las ontologías fueron obtenidas de la Web y abarcan diferentes dominios. Lamentablemente, no todas las ontologías presentan información sobre su naturaleza (dónde son utilizadas, quiénes las mantienen, etcétera).

A continuación, se presenta una breve descripción de las ontologías sobre las cuales se pudo encontrar información. El Cuadro 6.1 muestra el tamaño de la TBox de cada una de estas ontologías, y su expresividad.

- NCIT: Es un vocabulario para la atención clínica, actividades administrativas, información pública e investigación básica.
  - URL: `http://bioportal.bioontology.org/ontologies/NCIT`
- Go: El proyecto *Gene Ontology* provee vocabularios controlados sobre términos definidos representando propiedades sobre material bioquímico.
  - URL: `http://www.geneontology.org/`
- CTON: Es una ontología de tipos de células.
  - URL: `http://www.gong.manchester.ac.uk/CTON.html`
- FBbt XP: Es una ontología que representa la anatomía del *Drosophila melanogaster* (o como se la conoce comúnmente, *mosca de la fruta*).
  - URL: `http://www.obofoundry.org`
- Uberon: Es una ontología de anatomía para representar partes del cuerpo, órganos y tejidos de varias especies de animales.
  - URL: `http://uberon.github.io`
- Galen: Es una base de conocimiento de tipo biomédico.
  - URL: `http://bioportal.bioontology.org/ontologies/GALEN`
- Vicodi: Ontología creada para mejorar la comprensión del contenido digital de Internet.
  - URL: `http://www.vicodi.org/about.htm`

1 http://owl.cs.manchester.ac.uk/publications/supporting-material/owlcorpus/

- Lubm: Es una ontología artificial para medir el desempeño de razonadores OWL.
  - URL: http://ceur-ws.org/Vol-1014/paper_59.pdf
- Modlubm: Una versión modificada de la ontología LUBM.
  - URL: http://cgi.csc.liv.ac.uk/~frank/publ/filters.pdf
- Dolce: (*Descriptive Ontology for Linguistic and Cognitive Engineering*) es el primer módulo de la *WonderWeb Foundational Ontology Library*[2].
  - URL: http://www.loa.istc.cnr.it/old/DOLCE.html
- Wine: Una ontología para describir tipos y dominios de vinos, utilizada también para ejemplos y tutoriales.
  - URL: http://slidewiki.org/slide/24832
- Family: Es una ontología utilizada para varios tutoriales, ya que cuenta con una estructura intuitiva.
  - URL: http://wiki.csc.calpoly.edu/OntologyTutorial/
- Pizza: Otra ontología de *juguete* que se utiliza mucho en tutoriales.
  - URL: http://protegewiki.stanford.edu/wiki/Pr4_UG_ex_Pizza

Además de las ontologías presentadas en el Cuadro 6.1, también utilizamos otras ontologías para las cuales no pudimos obtener información. El Cuadro 6.2 presenta algunas características de las mismas.

6.2 RESULTADOS

Para correr los experimentos se utilizó una notebook con disco de estado sólido, 4 GB 1600 MHz DDR3 de memoria RAM, y un procesador 1,4 GHz Intel Core i5, corriendo en el sistema operativo macOS Sierra 10.12.3.

Los Cuadros 6.3 y 6.4 muestran los resultados obtenidos luego de correr las herramientas sobre las ontologías. Las columnas T_C, T_{DL2ML}, T_{SY4NCL}, T_{BLISS} y T_{TOT} representan los tiempos de carga de la ontología, de procesamiento en los módulos DL2ML, SY4NCL, BLISS y la suma de estos tiempos, respectivamente, medidos en segundos.

Cabe destacar que hay una de etapa más de post-procesamiento de datos (el MAPPER), pero los tiempos de esta etapa no fueron tenidos en cuenta dado que dicha etapa es realizada por scripts no optimizados.

Las columnas N_{GEN} y $|A|$ representan el número de generadores y el tamaño del grupo de automorfismos, respectivamente.

Como podemos observar en los Cuadros 6.3 y 6.4, todas las ontologías utilizadas en el marco de esta tesis tienen simetrías y los tiempos necesarios para detectarlas son pequeños, teniendo en cuenta el tamaño de las mismas.

2 http://cordis.europa.eu/project/rcn/60325_en.html

Ontología	|TBox|	Expresividad
ncit	46940	$\mathcal{ALC}$
go	104783	$\mathcal{ALC}$
cton	33160	$\mathcal{ALCF}$
fbbt xp	16014	$\mathcal{SHI}$
uberon	25683	$\mathcal{ALCQ}$
galen	4096	$(Horn) - \mathcal{SHIF}$
vicodi	213	$\mathcal{ALC}$
lubm	85	$\mathcal{ALC}$
modlubm	90	$\mathcal{ALC}$
dolce	780	$\mathcal{ALCFQC}$
wine	209	$\mathcal{ALCFQ}$
family	20	$\mathcal{ALCF}$
pizza	692	$\mathcal{ALCFQ}$

Cuadro 6.1: Resumen sobre las ontologías

Ontología	|TBox|	Expresividad
ged_p2	50488	$\mathcal{ALC}$
termdb	30374	$\mathcal{AL}$
protege	36205	$\mathcal{ALFI}$
module	22574	$\mathcal{ALC}$
1401	11858	$\mathcal{ALCFI}$
semintec	216	$\mathcal{ALCFI}$

Cuadro 6.2: Resumen sobre las otras ontologías

Ontología	Tiempo de ejecución					Tamaño del Grafo		N_{GEN}	$\|A\|$
	T_C	T_{DL2ML}	T_{SY4NCL}	T_{BLISS}	T_{TOT}	Nodos	Aristas		
ncit	8,563	1,747	2,66	32,833	37,24	164267	183694	10218	∞
go	16,081	17,653	6,002	34,275	57,93	432315	528394	8496	∞
cton	3,346	4,243	1,402	2,357	8,002	108724	125015	2278	6.47249e+1379
fbbt xp	2,743	2,744	1,066	1,289	5,099	75304	89447	1452	1.59369e+1112
uberon	3,537	3,828	1,598	0,763	6,189	107903	135288	818	2.58896e+492
galen	0,402	0,504	0,448	0,227	1,179	25262	30736	754	1.5284e+424
vicodi	3,139	3,664	0,03	0,016	3,71	976	1014	109	8.13478e+63
lubm	4,949	5,791	0,029	0,013	5,833	618	710	11	27648
modlubm	4,982	5,846	0,022	0,013	5,881	648	745	11	27648
dolce	0,149	0,229	0,124	0,023	0,376	6436	7835	8	1920
wine	1,877	2,39	0,042	0,013	2,445	1591	1879	7	2880
family	0,023	0,03	0,031	0,01	0,071	190	234	2	4
pizza	0,03	0,044	0,053	0,018	0,115	2898	4062	1	2

Cuadro 6.3: Resultados de las ontologías conocidas

Ontología	Tiempo de ejecución					Tamaño del Grafo		N_{GEN}	$\|A\|$
	T_C	T_{DL2ML}	T_{SY4NCL}	T_{BLISS}	T_{TOT}	Nodos	Aristas		
ged_p2	9,967	10,818	2,414	1286,457	1299,689	176639	192630	18180	∞
termdb	0,858	1,05	1,611	5,117	7,778	104136	114574	6316	∞
protege	3,214	2,912	4,459	11,714	19,085	251023	307063	5222	∞
module	2,139	2,619	0,385	1,241	4,245	29698	36361	3184	1.12116e+2713
1401	5,222	5,791	1,033	0,679	7,503	65256	88165	669	1.41773e+410
semintec	4,185	5,87	0,026	0,013	5,909	1139	1439	23	8.36076e+10

Cuadro 6.4: Resultados de las demás ontologías

Podríamos resumir los resultados del Cuadro 6.3 citando la conocida frase, atribuida al matemático griego Arquímedes de Siracusa, "¡Eureka!", y de hecho lo haremos: ¡Eureka! Pero también ampliaremos un poco más sobre lo obtenido.

Sobre el número de generadores

Fuimos capaces de encontrar generadores, en gran cantidad, en ontologías que son utilizadas actualmente por razonadores. No hay registro de que estas herramientas de inferencia (los demostradores) existentes exploten estos resultados para mejorar sus servicios.

Si bien es cierto que el número total de simetrías en las ontologías está relacionado con la estructura de la misma, no es posible estimar su número solo básandonos en la cantidad de axiomas con los que trabajemos, i.e., hay otros factores que influyen en la estructura del grafo sobre el cual se buscarán automorfismos: el tamaño de los axiomas, la cantidad de propiedades en común entre estos, etc.

Pero las pruebas empíricas realizadas hasta el momento parecen demostrar que en las ontologías existentes en la actualidad hay una alta posibilidad que encontremos una cantidad considerable de simetrías.

Sobre el tiempo de ejecución

El Cuadro 6.3 deja en evidencia que el enfoque propuesto en esta tesis, tanto teórico como en el desarrollo de las herramientas, es capaz de manejar $\mathcal{KB}$ reales con cientos de miles de axiomas sin mayores dificultades en cuanto al tiempo de ejecución ni consumo de memoria.

También hay que tener en cuenta que, como ya se mencionó, la búsqueda de simetrías puede realizarse *offline* y debe realizarse una sola vez por $\mathcal{KB}$.

El tiempo y espacio requeridos son verdaderamente insignificantes frente a la potencial ganancia, lo cual se convierte en una inversión sumamente rentable.

Otros análisis

Concentrémonos ahora en los resultados de las ontologías *ncit*, *cton* y *go*.

A simple vista no hay ninguna regla que pueda relacionar los valores que se observan entre sus columnas. Por ejemplo, podríamos pensar que mientras más axiomas tenga nuestra TBox, más generadores nuestra herramienta encontraría (pues nuestra fórmula modal contaría con más claúsulas y más símbolos proposicionales o relaciones); pero no es el caso, pues la ontología *go* cuenta con más axiomas que la ontología *ncit*, pero no cuenta con más generadores. También vemos este fenómeno entre las ontologías *ncit* y *cton*, la relación entre axiomas es de 1.4 a 1, mientras que la relación entre los generadores es de 4.5 a 1.

Otra observación importante a tener en cuenta es el tiempo de ejecución de T_{BLISS} con respecto al tamaño del grafo. El tamaño del grafo de la ontología *go* es unas 4 veces más grande que el de la ontología *cton*, pero los tiempos de corrida son de 17

a 1. Y entre *ncit* y *go* el grafo es casi de la mitad de tamaño, pero BLISS tarda solo dos segundos más en correr sobre el grafo de *go*.

No se han podido establecer relaciones muy directas entre los valores observados, solo se puede decir que dichos resultados dependen de la naturaleza intrínseca sobre la cual las ontologías están definidas. Quizá en algunas ontologías la mayoría de los conceptos están relacionados entre sí, mientras que en otras ontologías los conceptos se dividen más por secciones, es decir, bloques dentro de los cuales hay cierto tipo de conceptos, bajo ciertas relaciones específicas, y éstos no comparten ningún tipo de relación con conceptos de otras secciones o bloques.

Sin embargo, haber encontrado simetrías en las ontologías fue un resultado no trivial, ya que nada garantizaba que dichas simetrías vivieran dentro de las ontologías analizadas.

7 CONCLUSIONES Y TRABAJO FUTURO

El trabajo de esta tesis fue principalmente la investigación de simetrías en lógicas de descripción. Pero en el camino recorrido para llegar a tales resultados se han obtenido algunos otros resultados secundarios. Revisemos los detalles obtenidos.

7.1 RESUMEN DE LO HECHO

La principal tarea de esta tesis fue el estudio de las simetrías en lógicas de descripción. Como caso de estudio particular trabajamos directamente buscando simetrías en ontologías existentes y en uso en la Web.

Como es natural, en este trabajo comenzamos por los cimientos teóricos de las lógicas de descripción y la naturaleza de las ontologías. Dimos definiciones precisas de sintaxis y semántica de las lógicas de descripción, el porqué de OWL y cómo se pueden implementar bases de conocimientos expresadas en lógicas de descripción, en formato OWL.

Luego, dimos una introducción a las lógicas modales con su correspondiente sintaxis y semántica, unificamos la semántica de las lógicas de descripción y las lógicas modales y definimos una función de traducción Ψ para rescribir fórmulas de lógicas de descripción a fórmulas modales. El hecho de poder contar con una sola semántica para ambas lógicas y una función de traducción, nos permitió transferir todos los resultados teóricos obtenidos en el contexto de lógicas modales a lógicas de descripción necesarios para la detección de simetrías.

Vimos, bajo ciertas restricciones, que el problema de detección de simetrías era equivalente al de detección de automorfismos, por lo que definimos un algoritmo para construir un grafo a partir de una fórmula en CNF modal, y luego así explotar los automorfismos de dicho grafo.

También analizamos en detalle los módulos desarrollados para la detección de simetrías en ontologías, y dimos un ejemplo de uso muy simple para mostrar intuitivamente su funcionamiento.

Por último, probamos nuestra herramienta para detectar simetrías sobre ontologías existentes y en uso, obteniendo resultados totalmente positivos y alentadores: se encontraron simetrías en todas las ontologías con las que trabajamos.

7.2 QUÉ MÁS SE PUEDE HACER A PARTIR DE ACÁ

Sin lugar a dudas, haber detectado simetrías en las ontologías sobre las que trabajamos es un resultado remarcable. Dado que en [4] se demuestra que hacer uso de las simetrías encontradas tiene un impacto positivo en los sistemas implementados mejorando su rendimiento, y teniendo en cuenta que tanto estos sistemas como los razonadores que trabajan con ontologías están basados en tableaux, contamos con motivación suficiente para creer que hay una posibilidad de mejorar

los tiempos de ejecución de estos razonadores al incorporar las simetrías que se encuentren.

Sin embargo, todavía queda trabajo por hacer: solo se trabajó con la TBox de cada ontología, por lo que el siguiente paso es extender el dominio la función de traducción Ψ para que pueda trabajar con axiomas de la ABox. Es importante tener presente que traducir más axiomas puede tener un impacto en la cantidad de simetrías que encontramos, dado que podemos introducir axiomas que rompan ciertas simetrías.

BIBLIOGRAFÍA

[1] F. Aloul, A. Ramani, I. Markov y K. Sakallah. «Solving difficult instances of boolean satisfiability in the presence of symmetry». En: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2003), págs. 1117-1137.

[2] C. Areces. «The Case of Description and Hybrid Logics». Tesis doct. Universiteit van Amsterdam, 2000.

[3] C. Areces y D. Gorín. «Resolution with Order and Selection for Hybrid Logics». En: *Journal of Automated Reasoning* 46.1 (2011), págs. 1-42.

[4] C. Areces y E. Orbe. «Symmetry blocking». En: *Theoretical Computer Science* 606 (2015), págs. 25-41.

[5] F. Baader y B. Hollunder. «A terminological knowledge representation system with complete inference algorithm.» En: *Lecture Notes in Artificial Intelligence*. Ed. por Springer-Verlag. 567. Proceedings of the Workshop on Processing Declarative Knowledge (DPK-91). 1991, págs. 67-86.

[6] B. Benhamou, T. Nabhani, R. Ostrowski y M. Saidi. «Enhancing clause learning by symmetry in SAT solvers». En: *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*. Vol. 1. IEEE Computer Society, 2010, págs. 329-335.

[7] B. Benhamou y L. Sais. «Theoretical study of symmetries in propositional calculus and applications». En: *Automated Deduction—CADE-11: 11th International Conference on Automated Deduction Saratoga Springs, NY, USA, June 15–18, 1992 Proceedings*. Ed. por Deepak Kapur. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, págs. 281-294. ISBN: 978-3-540-47252-0. DOI: 10.1007/3-540-55602-8_172. URL: http://dx.doi.org/10.1007/3-540-55602-8_172.

[8] B. Benhamou y L. Sais. «Tractability through symmetries in propositional calculus». En: *Journal of Automated Reasoning* (1994), págs. 89-102.

[9] P. Blackburn, J. van Benthem y F. Wolter. *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. New York, NY, USA: Elsevier Science Inc., 2006. ISBN: 0444516905.

[10] A. Borgida, R. Brachman, D. McGuiness y L. Alperin Resnick. «CLASSIC: a structural data model for objects.» En: *Proceedings of the ACM SIGMOID International Conference of Management of Data*. 1989, págs. 59-67.

[11] R. Brachman. «What's in a concept: structural foundations for semantic networks.» En: *International Journal of Man-Machine Studies* (1977), 9:127-152.

[12] R. Brachman. «Associative Networks: The Representation and Use of Knowledge by Computers». En: ed. por Nicholas. Findler. Academic Press, Inc., 1979. Cap. On the epistemological status of semantic networks. Págs. 3-50.

[13] R. Brachman, V. Gilbert y H. Levesque. «An essential hybrid reasoning system: knowledge and symbol level accounts in KRIPTON.» En: *Proceedings Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-85)*. 1985, págs. 532-539.

[14] R. Brachman y H. Lavesque. «The Tractability of Subsumption in Frame-Based Description Languages». En: *AAAI'84 Proceedings of the Fourth AAAI Conference on Artificial Intelligence* (1984), págs. 34-37.

[15] R. Brachman y J. Schmolze. «An overview of the KL-ONE knowledge representation system.» En: *Cognitive Science* (1985), 9(2):171-216.

[16] C. Brown, L. Finkelstein y P. Purdom Jr. «Backtrack searching in the presence of symmetry». En: *Applied algebra, algebraic algorithms and error-correcting codes*. Springer, 1989, págs. 99-110.

[17] V. Chaudhri, A. Farquhar, R. Fikes, P. Karp y J. Rice. «OKBC: A programmatic foundation for knowledge base interoperability». En: *Proceedings of the 15th National Conference on Artificial Intelligence*. 1998, págs. 600-607.

[18] E. Clarke, R. Enders, T. Filkorn y S. Jha. «Exploiting symmetry in temporal logic model checking». En: *Formal Methods in System Design* 9.1 (1996), págs. 77-104.

[19] M. Cohen, M. Dam, A. Lomuscio y H. Qu. «A symmetry reduction technique for model checking temporal-epistemic logic». En: *IJCAI* 9 (2009), págs. 721-726.

[20] J. Crawford. «A theorical analysis of reasoning by symmetry in first-order logic.» En: *Proceedings of AAAI Workship on Tractable Reasoning*. 1992, págs. 17-22.

[21] C. Cubadda. «Variantes de l'algorithmes de sl-résolution avec retenue d'informations». Tesis doct. Gia Luminy (Marseille), 1998.

[22] M. Dean, G. Schreiber, F. Bechhofer S. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuiness, P.F. Patel-Schneider y L.A. Stein. *OWL web ontology language reference. W3C Recommendation.*

[23] A. Donaldson. «Automatic techniques for detecting and exploiting symmetry in model checking». Tesis doct. University of Glasgow, 2007.

[24] A. Donaldson y A. Miller. «Automatic Symmetry Detection for Model Checking Using Computational Group Theory». En: *FM 2005: Formal Methods: International Symposium of Formal Methods Europe, Newcastle, UK, July 18-22, 2005. Proceedings*. Ed. por John Fitzgerald, Ian J. Hayes y Andrzej Tarlecki. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, págs. 481-496. ISBN: 978-3-540-31714-2. DOI: 10.1007/11526841_32. URL: http://dx.doi.org/10.1007/11526841_32.

[25] F. Donini. *Foundations of Knowledge Representation and Reasoning*. Ed. por G. Lakemeyer y B. Nebel. Springer, 1994.

[26] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness y P. Patel-Schneider. «OIL: An Ontology Infrastructure for the Semantic Web». En: *IEEE Intelligent Systems* 16.2 (mar. de 2001), págs. 38-45. ISSN: 1541-1672. DOI: 10.1109/5254.920598. URL: http://dx.doi.org/10.1109/5254.920598.

[27] D. Fensel, W. Wahlster y H. Lieberman. «Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential». En: MIT press, 2002. Cap. DAML-ONT: An Ontology Language for the Semantic Web.

[28] J. Fraleigh y V. Katz. *A first course in abstract algebra*. Addison-Wesley, 2003.

[29] W. Grosso, H. Eriksson, R. Fergerson, J. Gennari, S. Tu y M. Musen. «Knowledge Modeling at the Millennium (The Design and Evolution of Protégé-2000)». En: *Proceedings Proceedings of the Knowledge Acqusition Workshop*. 1999, págs. 16-21.

[30] T. Gruber. «Toward Principles for the Design of Ontologies Used for Knowledge Sharing». En: *International Journal Human-Computer Studies* (1995), 43(5-6):907-928.

[31] *Haskell*. URL: https://www.haskell.org/.

[32] P. Hayes. *Formal Theories of the Common-Sense World*. Ed. por J. Hobbs y R. Moore. Vol. The Second Naive Physics Manifesto. Intellect Ltd, 1985.

[33] J. Heflin y J. Hendler. «Dynamic ontologies on the Web». En: *Proceedings of the Seventeenth National Conference on Artificial Intelligence*. 2000, págs. 443-449.

[34] G. Hoffmann. «HTab: a Terminating Tableaux System for Hybrid Logic». Tesis doct. INRIA Lorraine, 2009.

[35] I. Horrocks. «DAM+OIL: a Description Logic for the Semantic Web». En: *IEEE Data Engineering Bulletin* 25 (2002), págs. 4-9.

[36] I. Horrocks y P. Patel-Schneider. «Reducing OWL entailment to description logic satisfiability». En: *Web Semantics: Science, Services and Agents on the World Wide Web* 1.4 (2004). International Semantic Web Conference 2003, págs. 345-357. ISSN: 1570-8268. URL: http://www.sciencedirect.com/science/article/pii/S1570826804000095.

[37] T. Junttila y P. Kaski. «Engineering an efficient canonical labeling tool for large and sparse graphs». En: *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments*. Ed. por David Applegate, Gerth Stolting Brodal, Daniel Panario y Robert Sedgewick. New Orleans, LA: SIAM (Society for Industrial y Applied Mathematics), ene. de 2007, págs. 135-149.

[38] H. Katebi, A. Sakallah y L. Markov. *Theory and Applications of Satisfiability Testing*. Vol. 6175. Springer-Verlag Berlin Heidelberg, 2010, págs. 113-127.

[39] B. Krishnamurthy. «Short proofs for tricky formulas». En: *Acta Informatica* (1985), 22(3):253-275.

[40] R. MacGregor y R. Bates. *The Loom knowledge representation language.* Inf. téc. Information Science Institute, University of Southern California, Marina del Rey, California., 1987.

[41] A. Miller, A. Donaldson y M. Calder. «Symmetry in Temporal Logic Model Checking». En: *ACM Comput. Surv.* 38.3 (sep. de 2006). ISSN: 0360-0300. DOI: 10.1145/1132960.1132962. URL: http://doi.acm.org/10.1145/1132960.1132962.

[42] M. Minsky. *A Framework for Representing Knowledge*. Inf. téc. Massachusetts Institute of Technology Cambridge, 1981.

[43] D. Nardi y R. Brachman. *The Description Logic Handbook: Theory, Implementation, and Applications*. Ed. por Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi y Peter F. Patel-Schneider. Cambridge University Press, 2003.

[44] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator y W. Swartout. «Enabling technology for knowledge sharing». En: *AI Magazine* (1991), 12(3):16-36.

[45] E. Orbe. «Simetrías en Razonamiento Automático». Tesis doct. FaMAF, Universidad Nacional de Córdoba, Argentina, 2014.

[46] L. Oxusoff. «L'évaluation sémantique en calcul propositionnel». Tesis doct. Université Aix-Marseille 2, 1989.

[47] F. Patel-Schneider, J. Hayes e I. Horrocks. *OWL web ontology language semantics and abstract syntax. W3C Recommendation.* URL: http://www.w3.org/TR/owl-semantics.

[48] J. Quantz y C. Kindermann. *Implementation of the BACK system, version 4*. Inf. téc. FB Informatik, Technische Universität Berlin, Berlin, Germany., 1990.

[49] M. Quillian. «Semantic Information Processing». En: ed. por Marvin Minski. MIT press, 1968. Cap. Semantic Memory Models, págs. 227-270.

[50] *RDF*. URL: https://www.w3.org/RDF/.

[51] *RDFS*. URL: https://www.w3.org/TR/rdf-schema/.

[52] Giovanni Rescia. *Thesis*. URL: https://github.com/giovannirescia/thesis.

[53] *SAUCY*. URL: http://vlsicad.eecs.umich.edu/BK/SAUCY/.

[54] *Scala*. URL: https://www.scala-lang.org.

[55] K. Schild. «A correspondence theory for terminological logics.» En: *IJCAI'91 Proceedings of the 12th international joint conference on Artificial intelligence* 1 (1991), págs. 466-471.

[56] *Scowl*. URL: http://github.com/phenoscape/scowl.

[57] A. Seress. «An introduction to computational group theory». En: *Notices of the AMS* 44 (1997), 44:671-679.

[58] *Unicode*. 2016. URL: http://www.unicode.org/.

[59] *URI*. URL: https://www.w3.org/TR/uri-clarification/.

[60] W3C. *OWL API*. URL: http://owlapi.sourceforge.net/.

[61] *WordNet*. URL: https://wordnet.princeton.edu/.

[62] *XML*. URL: https://www.w3.org/XML/.