Universidad Nacional del Sur

Tesis de Doctor en Ciencias de la Computación

Consolidación de Ontologías Datalog$^{\pm}$

Cristhian Ariel David Deagustini

Bahía Blanca Argentina

2015

Prefacio

Esta Tesis se presenta como parte de los requisitos para optar al grado Académico de Doctor en Ciencias de la Computación, de la Universidad Nacional del Sur y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otra. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el ámbito del Departamento de Ciencias e Ingeniería de la Computación durante el período comprendido entre el 5 de julio de 2011 y el 25 de noviembre de 2015, bajo la dirección del Dr. Guillermo R. Simari, Universidad Nacional del Sur y del Dr. Marcelo A. Falappa, Universidad Nacional del Sur.

.....

Cristhian Ariel David Deagustini
cadd@cs.uns.edu.ar
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca, 25 de noviembre de 2015

UNIVERSIDAD NACIONAL DEL SUR
Secretaría General de Posgrado y Educación Continua

La presente tesis ha sido aprobada el .../.../..., mereciendo la calificación de .....(.....)

Agradecimientos

Quisiera agradecer a todos aquellos que han colaborado de alguna o otra forma a que haya logrado culminar mi doctorado. En primer lugar, infinitas gracias a mis directores, Guillermo y Marcelo, por toda la paciencia que han mostrado durante estos años, especialmente al comienzo esta aventura donde realmente necesité mucho apoyo, y ustedes supieron con dedicación y sabiduría orientarme para que definiéramos objetivos, y luego lográramos alcanzarlos. Especialmente quiero agradecer el hecho de que me hayan brindado su confianza a pesar de venir de una universidad diferente y de no conocerme ustedes, con los riesgos que eso suponía. Espero haber estado a la altura. No alcanzan las palabras para expresar mi gratitud por todas las consideraciones que han tenido para conmigo a través de los años, y por el hecho de dar permanentemente un esfuerzo extra, yendo siempre mucho más allá de sus obligaciones como directores.

También quiero agradecer de manera especial a mis "sub-directores", Vani y Gotti. Ustedes tienen muchísimo que ver con que todo el esfuerzo de estos años haya llegado a buen puerto. Gotti, siempre te voy a estar agradecido por todas esas charlas que hemos tenido a lo largo de estos años, donde realmente he aprendido muchísimo acerca de lo que implica estar en el mundo de la investigación. Yo me había metido en esto sin saber por completo que era lo que me esperaba, y todos tus consejos me fueron de gran ayuda, no sólo a la hora de discutir cosas técnicas de algún trabajo, sino principalmente cuando hablábamos del mundo de la investigación en general, que disfruté mucho. Sin duda que acorté años de aprendizaje a la fuerza simplemente con escucharte contarnos y explicarnos tus experiencias. Mil gracias! Y respecto a vos Vani, realmente no puedo decirte cuanto agradezco todos estos años de trabajo. Lo he dicho muchas veces a quien quiso oírlo, mi tesis no estaría aquí ahora ni sería lo que es si no fuera por todo lo que me has ayudado. Estoy seguro que el gran punto de inflexión en mi carrera en el mundo de la investigación siempre va a ser el momento en que estábamos reunidos con Guillermo y

Marcelo y ellos sugirieron que te pida unirte a nuestro grupo. Y por supuesto, el hecho que hayas aceptado, y que hayas aportado TANTO para el desarrollo de nuestros trabajos, y de mí como investigador. Muchísimas gracias por todos estos años de aguantar mis molestias, mis mails de tamaño descomunal a deshoras, mis consultas (las inteligentes y de las otras también jaja), por todas las conferencias y reuniones donde sin duda avanzaba más en un par de horas de trabajo que en el resto de la semana, y por sobre todo por estar siempre dispuesta a darme una mano. Muchas gracias por todo!

Quiero agradecer también a todos los chicos de la salita de becarios: Santiago, Carlitos, Maxi, Harry, Juli, Noni, Gotti, Kcho, Anita, Ceci, Jime, Fabio, Diego, Lucho, Mauro y Martin. Un grupo de personas excepcional, siempre dispuestas a ayudar y que realmente hacen que disfrute enormemente mi trabajo. Muchas gracias por todas las charlas (técnicas, y también de las otras) que realmente amenizaron el esfuerzo todos estos años. En especial quiero agradecer a los energúmenos de siempre: Santi, Carlitos, Harry, Maxi y también Manu (a.k.a. el desertor). Muchísimas gracias viejos por todo el aguante, las charlas, los partidos de fútbol, asados y truco, partidas de póker, y tantas cosas que hemos vivido a través de estos años. Sin duda son de las cosas más importantes que me llevo de esta experiencia.

También quiero aprovechar la oportunidad para agradecer a todos en el Departamento de Ciencias e Ingeniería de la Computación, que en estos años me han hecho sentir como si hubiera estado toda la vida en esta universidad a pesar del hecho de venir de otro lado y no conocerme. Por otro lado también agradezco a los profesores de mi universidad, la Universidad Nacional de Entre Ríos, por todos los años de dedicación y esfuerzo durante mi carrera allí. Es indudable que tienen mucho que ver con que haya llegado a este punto, y realmente agradezco enormemente la formación que me brindaron. En especial quiero agradecer a mi profesor de Inteligencia Artificial (entre otras cátedras) Cristian Pacífico. Muchísimas gracias Cristian por todo: por esa charla en el final de Estructura de Datos (que sin duda definió mi carrera de Licenciatura, y más), por estar siempre dispuesto a darme una mano, y principalmente por haberme ayudado a dar mis primeros pasos en el Doctorado. Evidentemente no estaría aquí ahora si no hubiera sido por todo tu esfuerzo desde el momento en que te comenté que pensaba seguir una carrera en el mundo de la investigación. Eternamente agradecido!

Tampoco puedo olvidarme de agradecer al Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), que a través de los años han confiado en mi, fi-

nanciando por completo mi Doctorado través de diversas becas, lo que brindo el sustento necesario para que esto pueda ser llevado a cabo.

Para concluir, quisiera dedicar la tesis y dar los agradecimientos más importantes de todos: a mi familia. Primeramente, quiero agradecer al amor de mi vida Alicia. Muchísimas gracias mi amor por todo el apoyo durante estos años, que incluyeron bancarte estar alejada de nuestras familias sólo por hacerme el aguante en esta aventura, sin dudas que esta tesis es mitad tuya (y sí, la mitad buena). Y millones de gracias por estar siempre ahí, por brindarme todo lo necesario para que yo pueda hacer mi trabajo, y sobre todo por darme a esos dos soles hermosos que tenemos de hijos. Dante y Amaia, esto también es por ustedes!! No creo que lleguen a darse una dimensión de lo feliz que me hacen los tres!! Sin ustedes nada de esto hubiera sido posible!!

Quiero agradecer también a mis hermanos por toda esta vida juntos, y por estar ahí siempre. Y finalmente quiero agradecerlos a mis padres, Jorge y Roxana. Viejos, quiero que sepan que esto también es de ustedes. Sin duda alguna las enseñanzas que me han dado en la vida son mucho más importantes que cualquier educación formal que haya tenido. Siempre que me chocaba una pared y pensaba que no podía seguir los recordaba y me decía: ¿sí ellos nunca bajaron los brazos, cómo voy a abandonar yo? No tenía derecho a desperdiciar todo el esfuerzo que hicieron en su vida para darme a mí y mis hermanos todo lo que necesitamos, a base de incontables horas de trabajo y dedicación. Era imposible que teniendo ese ejemplo tan a mano pensara en rendirme, no hubiera sido justo. Así que les repito, esto también es de ustedes.

Muchas gracias a todos!

Ariel

PD: no puedo concluir estos agradecimientos sin mencionar a otra parte muy importante en mi vida, el Club Atlético Boca Juniors, que me ha brindado incontables alegrías y algunas tristezas, pero siempre me dio ese escape que necesitaba para despejar la mente y renovar fuerzas.

Resumen

En la presente tesis nos enfocamos en el manejo de dos problemas diferentes pero relacionados que suelen aparecer en el conocimiento, especialmente en entornos colaborativos: inconsistencias e incoherencias. Inconsistencia es un problema clásico y ampliamente reconocido en la representación de conocimiento, el cual trae importantes consecuencias para los mecanismos clásicos de inferencia. Incoherencia, por otra parte, ha recibido cada vez más atención desde el surgimiento de lenguajes ontológicos; la misma se relaciona con conflictos en el conjunto de reglas ontológicas que hacen a tales reglas imposibles de satisfacer al mismo tiempo. En este trabajo formalizamos la noción de incoherencia en ontologías Datalog$^{\pm}$, que se encontraba ausente en la literatura, en términos de la satisfacibilidad del conjunto de restricciones en las mismas, y mostramos como bajo ciertas condiciones incoherencia puede llevar a ontologías Datalog$^{\pm}$ inconsistentes.

La contribución principal de este trabajo es el desarrollo de dos operadores noveles para la restauración tanto de la consistencia como la coherencia en ontologías Datalog$^{\pm}$. Los enfoques propuestos se basan en kernel contraction. En el primero de ellos la restauración se realiza mediante la aplicación de funciones de incisión que seleccionan fórmulas para remoción de los conjuntos incoherentes/inconsistentes mínimos encontrados en las ontologías. Tal operador trata los conflictos mínimos de manera local, sin tener en cuenta la relación (si es que existe) entre los diferentes conflictos mínimos. El otro enfoque, que puede ser visto como un enfoque global, tiene en cuenta tal relación mediante el agrupamiento de conflictos mínimos relacionados en nuevas estructuras llamadas clusters, mediante el uso de una relación de solapamiento.

En esta disertación presentamos construcciones tanto para el enfoque local como el global, junto con las propiedades que se espera que los mismos satisfagan, expresadas a través de postulados. Finalmente, establecemos la relación entre las construcciones y las propiedades mediante el uso de teoremas de representación. Si bien la propuesta esta enfocada en la consolidación de ontologías Datalog$^{\pm}$, estos operadores pueden ser aplicados a otros tipos de lenguajes ontológicos, tales como las Lógicas Descriptivas, haciéndolos aptos para su uso en ambientes colaborativos como la Web Semántica.

Abstract

In this thesis we focus on the management of two different but related problems that arise in knowledge, especially in collaborative environments: inconsistency and incoherence. Inconsistency is a classic, widely acknowledged issue in knowledge representation, with important consequences to classical reasoning mechanisms. Incoherence, on the other hand, has been increasingly received attention since the arousal of ontological languages; it is related to conflicts in the set of ontological rules that make such rules impossible to satisfy at the same time. In this work we formalize the notion of incoherence for Datalog$^{\pm}$ ontologies, which was previously absent in the literature, in terms of satisfiability of sets of constraints, and show how under specific conditions incoherence leads to inconsistent Datalog$^{\pm}$ ontologies.

The main contribution of this work is the development of two novel approaches to restore both consistency and coherence in Datalog$^{\pm}$ ontologies. The proposed approaches are based on kernel contraction. In the first one restoration is performed by the application of incision functions that select formulæ to delete from the minimal incoherent/inconsistent sets encountered in the ontologies. Such operator treats minimal conflicts in a local manner, disregarding the relation (if any) among different minimal conflicts. The other approach, which can be seen as a global one, does acknowledge such relation by means of the grouping of related minimal conflicts in new structures called clusters, by means of an overlapping relation.

We present constructions for both local and global consolidation operators, along with the properties expected to be satisfied by them, expressed through postulates. Finally, we establish the relation between the constructions and the properties by means of a representation theorems. Although this proposal is presented for Datalog$^{\pm}$ ontologies consolidation, these operators can be applied to other types of ontological languages, such as Description Logics, making them apt to be used in collaborative environments like the Semantic Web.

Índice general

Capítulo 1

Introducción

1.1. Motivación: importancia del manejo de inconsistencias e incoherencias

En tiempos donde la colaboración e interacción entre sistemas se ha vuelto moneda corriente es cada vez más frecuente el encontrarse con problemas derivados de tal interacción, apareciendo a su vez desafiantes oportunidades de investigación asociadas a los mismos. En particular, el manejo de información conflictiva es un importante problema que debe ser atacado [GCS10, HvHH$^+$05, HvHtT05, BQL07], especialmente cuando se integra conocimiento proveniente de diferentes fuentes [BHP09, BKM91, AK05], o cuando tal conocimiento será explotado por procesos de razonamiento automáticos. El más conocido dentro de los conflictos en información es el de la inconsistencia. Inconsistencia es un concepto clásico dentro de la Representación de Conocimiento y el Razonamiento, y se refiere a teorías tales que no es posible encontrar para ellas un modelo.[1]

Ejemplo 1.1 (Inconsistencia) *Considere las siguientes afirmaciones que brinda una persona cuando le preguntamos acerca de sus recientes vacaciones.*

"En mis vacaciones pasé por Mar Del Plata, una ciudad de Argentina. Muy lindo lugar, aunque por desgracia me tocó un día lluvioso. [...] La verdad es que fueron unas muy buenas vacaciones. En particular, lo mejor de mi paso por Argentina es que todos los días pude salir de excursión, no me ha tocado ningún día de lluvia."

[1] En el presente trabajo se asume que el lector posee conocimientos básicos de Lógica.

Aquí se presenta una inconsistencia, ya que no puede ser verdadero al mismo tiempo que la persona haya estado de vacaciones en Mar del Plata, Argentina (p) en un día lluvioso (q) siendo que todos los días que estuvo de vacaciones en Argentina no llovió ($p \rightarrow \neg q$).

La importancia del manejo de la inconsistencia surge del problema que la misma acarrea en las relaciones de consecuencia clásicas: las mismas son *explosivas* [HvHtT05, GCS10], en el sentido de que cualquier fórmula es consecuencia de una contradicción lógica, pudiendo por lo tanto derivar el lenguaje completo a partir de una base de conocimiento inconsistente. Claramente, las conclusiones basadas en conocimiento inconsistente obtenidas a través de tales procesos de inferencia pueden llegar a ser completamente irrelevantes, ya que no sólo podremos inferir algo sino también su contradicción, sin poder determinar cual de ambas fórmulas es realmente válida. Sin ir más lejos, esto se hace evidente en la situación explicada en el Ejemplo 1.1; a partir de ese conocimiento inconsistente podemos inferir tanto que la persona en cuestión tuvo un día lluvioso en sus vacaciones por Argentina como que no, y no podemos decidir cual de las dos afirmaciones es la valedera. De esta manera se evidencia la gran importancia del manejo de la inconsistencia en lógicas como la Lógica Proposicional o la Lógica de Predicados. Es más, esta situación es también observable en contextos de conocimiento ontológico (aquél donde separamos los hechos conocidos acerca del mundo de las reglas ontológicas utilizadas para la obtención de conocimiento implícito a partir de estos hechos). En tales entornos las relaciones de consecuencia, si bien se encuentran adaptadas a la distinción hecha entre hechos y reglas, siguen en muchos casos íntimamente relacionadas con la consecuencia clásica, compartiendo muchas de sus características, entre ellas la falta de tolerancia a conocimiento inconsistente.

En adición a la inconsistencia, en entornos ontológicos podemos encontrar otro tipo de conflictos, relacionados a un fenómeno conectado en cierta forma con inconsistencia pero a su vez con sus características propias: incoherencia [FHP$^+$06, Bor95, BB97, KPSH05, SHCvH07, QH07], un concepto que ha surgido con el advenimiento de entornos ontológicos como Datalog$^\pm$. Tal fenómeno versa en una característica del conocimiento expresado a través de las reglas que hace que el mismo no pueda ser aplicado sin generar problemas de consistencia (es decir, un conjunto de reglas las cuales no pueden ser aplicadas sin violar *inevitablemente* alguna de las restricciones impuestas al conocimiento, haciéndolas por lo tanto insatisfacibles). Intuitivamente podemos ver a la incoherencia como una inconsistencia latente o potencial; esto es, el conocimiento incoherente no representa una violación de

consistencia en sí mismo, pero cuando un conocimiento ontológico (un conjunto de reglas) incoherente es considerado junto con hechos relevantes (hechos que activen las reglas en cuestión) entonces la violación es inevitable. Esto marca la diferencia más grande con el concepto de inconsistencia: la incoherencia *no necesariamente* implica una contradicción; al punto de que *incoherencia no puede provocar una violación explícita* sin ser considerada junto con instanciaciones de las reglas. Es decir, mientras que la inconsistencia es un conflicto puesto de manifiesto (explícito, una violación que ya está ocurriendo) la incoherencia se refiere a conflictos latentes (implícitos, pero que no necesariamente estén ocurriendo en el momento); se podría pensar por lo tanto en la incoherencia como un preludio de la inconsistencia. Esta diferencia se vuelve más evidente si comparamos cómo se resuelven los problemas de incoherencia y cómo los de inconsistencia: mientras que para solucionar los primeros debemos eliminar/modificar las reglas utilizadas para inferir conocimiento, para los segundos debemos, en general, modificar los datos. Es importante aclarar que en Lógica Proposicional incoherencia e inconsistencia se vuelven indistinguibles, debido a que no hay una distinción entre hechos y reglas (*esto es,* no existe conocimiento ontológico), ya que todo conocimiento es expresado como una proposición. Sin embargo, para conocimiento ontológico este fenómeno reviste gran importancia, ya que, como veremos en la presente tesis, el no considerarlo puede ocasionar problemas graves en la definición de procesos de manejo de inconsistencia.

Ejemplo 1.2 (Incoherencia) *Considere el siguiente ejemplo de conocimiento incoherente.*

"Todos los cantantes de rock cantan a un alto volumen. Este hecho, el cantar a un volumen alto, hace que la persona se lastime la garganta. [...] Los cantantes de rock evidentemente son personas que tienen la capacidad de cantar, por algo en sus bandas los han elegido. Y sabemos que si alguien puede cantar entonces no tiene la garganta lastimada."

Si bien claramente hay un problema con el conocimiento expresado, este no es inconsistente en sí mismo, ya que no estamos hablando de ningún cantante. El problema radica cuando queremos considerar algún cantante particular, por ejemplo al decir que Axl es un cantante de rock. A partir de ahí con el conocimiento que tenemos podemos inferir que Axl debe tener la garganta lastimada y que tiene la capacidad de cantar, algo que se presupone no puede pasar. Esta situación se verifica para cualquier cantante que queramos considerar, y por lo tanto podemos decir que es imposible satisfacer al mismo tiempo todas las reglas sobre el mundo de los cantantes de rock expresadas anteriormente.

Es decir, ese conocimiento es incoherente: al considerar el conocimiento ontológico (las reglas expresadas) junto con una instancia particular (hecho) cualquiera el mismo se torna inconsistente.

En el Ejemplo 1.2 podemos notar más claramente como, en entornos ontológicos, la incoherencia es un conflicto latente. Como se explicó, las sentencias no constituyen en si mismas un conflicto explícito (no hay una violación particular). Sin embargo, claramente ese conocimiento es una violación *en potencia*; esto es, al considerarlo junto con una instancia particular cualquiera (en este caso *cualquier* cantante de rock) la violación inevitablemente se hace explícita (es decir, se llega a inconsistencia). Es por esto que hablamos de la incoherencia como un preludio de la inconsistencia, ya que *podemos considerar a la incoherencia como un conflicto esperando a suceder*. Esta transformación entre conflictos latentes y explícitos, *esto es,* el pasaje de incoherencia a inconsistencia al considerar instancias particulares junto con conocimiento incoherente, se muestra en la Figura 1.1.

Por lo recientemente expresado es que en la Inteligencia Artificial (IA) muchos trabajos se han enfocado en el manejo de los problemas de inconsistencia (y también, aunque en menor medida, incoherencia) en conocimiento de forma tal que se evite de alguna manera los problemas acarreados por la misma. Básicamente podemos dividir a estos esfuerzos en dos grandes grupos:

- Aquellos esfuerzos que se centran en modificar de alguna forma el conocimiento inconsistente de forma tal que la consistencia en el mismo sea recuperada, para luego aplicar relaciones de consecuencia clásicas sobre el conocimiento consistente obtenido (generalmente subconjuntos maximalmente consistentes de la base de conocimiento original). De esta forma, si bien la base de conocimiento original es inconsistente y por lo tanto capaz de derivar todo el lenguaje, al utilizar una modificación consistente de la misma se pueden aplicar relaciones de inferencia clásicas de manera segura. Este es uno de los objetivos perseguidos por la teoría de *Revisión de Creencias* [Gär03, GR92, AGM85, Dal88, QLB06, Han97a, Han94]. Por ejemplo, volviendo a la situación expresada en el Ejemplo 1.1, podríamos elegir descartar la afirmación de que la persona estuvo en un día lluvioso en Mar del Plata (q). Luego, el razonamiento sobre el conjunto $\{p, p \rightarrow \neg q\}$ puede efectuarse sin problemas mediante una relación de inferencia clásica.

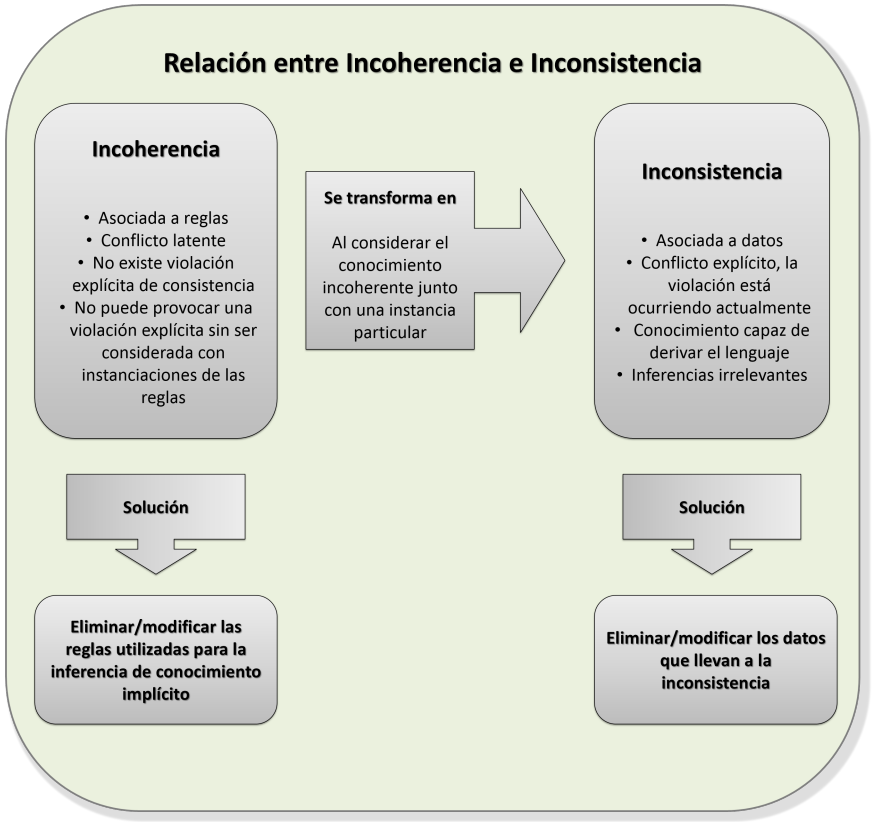

Figura 1.1: Relación entre Incoherencia e Inconsistencia: transformación entre conflicto latente y conflicto explícito

- Alternativamente, se puede modificar cómo el conocimiento es inferido mediante la definición de relaciones de inferencia alternativas, estableciendo de esta forma métodos de razonamiento tolerantes a inconsistencias, que son aquellos que pueden obtener conclusiones consistentes a partir de conocimiento inconsistente. Entre los ejemplos más importantes de tales métodos podemos citar a la *Argumentación* y el *Razonamiento Rebatible* para conocimiento expresado en formalismos como Lógica Proposicional y afines [Sim89, SL92, DW09, GS04, MWF10, Nut88, Pol87, Pra10], así como también la Resolución Consistente de Consultas (Consistent Query Answering - CQA) y métodos derivados de la misma para aplicación en bases de datos relacionales y otros lenguajes expresivos utilizados en tecnologías como Web Semántica [ABC99, LMS12, LLR$^+$10]. En el Ejemplo 1.1, en Argumentación podríamos sopesar las razones que tenemos para creer que la persona estuvo en Mar del Plata en un día lluvioso (q) contra aquellas que tenemos para creer que no llovió cuando estuvo en dicha ciudad ($\neg q$). Si por ejemplo preferimos elegir creer que es más plau-

sible que dicha persona se haya equivocado al afirmar que salió de excursión todos los días en Argentina porque no hubo día lluvioso, entonces la inferencia de $\neg q$ a través del conjunto $\{p, p \rightarrow \neg q\}$ queda descartada y nuestro método de razonamiento afirmará que la persona en cuestión sí estuvo en un día lluvioso en Argentina (q).

En la presente tesis nos enfocamos en técnicas encuadradas dentro del primer grupo descrito, es decir aplicaremos un enfoque de modificación de ontologías incoherentes e inconsistentes para resolver los conflictos en las mismas, aplicando para ello técnicas de Revisión de Creencias.

1.2. La revisión de creencias en IA

El principal objetivo de la Revisión de Creencias, también conocida como *Teoría de Cambio*, es tratar de modelar la dinámica del conocimiento. Es decir, como el conocimiento cambia para adaptarse a influencias externas, *esto es,* después de recibir cierta información externa. Si bien establecer el origen de ideas es una tarea difícil (y frecuentemente controversial), puede encontrarse a los orígenes de la Teoría de Cambio en el trabajo de Isaac Levi [Lev77], quien discutió primeramente los problemas concernientes a este campo de investigación, y a la propuesta de William Harper de interrelacionar operadores de cambio de una forma racional [Har75]. Sin embargo, los principales avances del campo surgieron durante la década de 1980 cuando Carlos Alchourrón y David Makinson estudiaron cambios en códigos legales [AM81], y Peter Gärdenfors introdujo postulados racionales para operadores de cambio [Gär82]. Luego, los tres autores publicaron un trabajo fundamental de la teoría de cambio que pasó a ser conocido como el modelo AGM [AGM85], denominado de esta forma debido a las iniciales de los tres autores. Dicho modelo proveyó el marco de trabajo (framework) formal más general hasta ese momento para estudios de cambio de creencias y representa un punto de quiebre en la evolución de la teoría de cambio de creencias. No sólo se trató de uno de los primeros y más importantes desarrollos en cuanto a operadores de cambio de creencias en sí, sino que también el trabajo de Alchourrón, Makinson y Gärdenfors sentó las bases de un "estilo" en la presentación de resultados en el área, mediante la utilización de los teoremas de representación provistos (también llamados caracterizaciones axiomáticas) de la contracción y la revisión, los cuales sirven para brindar una clara conexión entre las construcciones

de los operadores y las propiedades esperadas de los mismos. Tales teoremas se basan en mostrar la relación entre dos componentes fundamentales de la teoría de cambio de creencias: los postulados y las construcciones de operadores (Figura 1.2). Los postulados de racionalidad [AGM85, Gär88] determinan restricciones que los operadores deben satisfacer. Ellos tratan a los operadores como cajas negras, describiendo sus comportamientos con respecto a la entrada en casos básicos, pero no los mecanismos internos usados. Por otro lado las construcciones se refieren a la contraparte de los postulados, es decir, a definir como conseguir el objetivo esperado por parte del operador. El impacto de AGM en la teoría de cambio de creencias fue enorme; es al día de hoy que este trabajo es considerado la piedra basal a partir de la cual toda la Teoría de Cambio evolucionó.

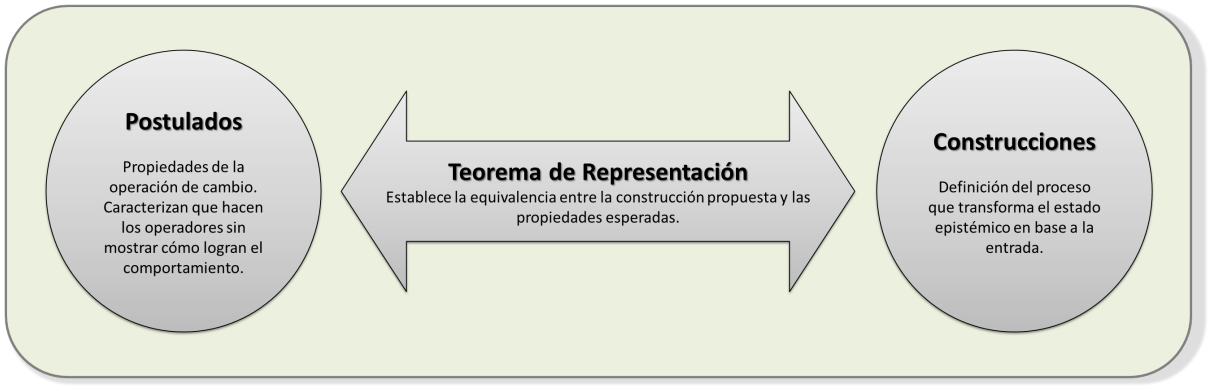

Figura 1.2: Modelo AGM: usando Teoremas de Representación para establecer correspondencias entre construcciones y propiedades de operaciones de cambio.

Desde la introducción del modelo AGM, diferentes frameworks para la dinámica de conocimiento han sido propuestos, junto con sus respectivos modelos epistémicos (el formalismo en el cual las creencias o el conocimiento son representadas y en el cual diferentes clases de operadores pueden ser definidos). La gran mayoría de estos trabajos versan sobre alguna de las operaciones básicas de cambio definidas en AGM, que presentamos a continuación.

- **Expansión**: el resultado de expandir una base de conocimiento mediante una sentencia es una nueva base de conocimiento, usualmente más grande, que infiere a la sentencia en cuestión. Intuitivamente, la nueva sentencia (que se espera sea consistente con el conocimiento actual) es directamente agregada a la base de conocimiento.

- **Contracción**: el resultado de contraer de la base de conocimiento una fórmula es una nueva base de conocimiento que usualmente no infiere a la fórmula contraída (intuitivamente, la única excepción a esto es si queremos contraer una tautología $\top$, ya que la misma siempre podrá ser inferida de la base de conocimiento resultante).
- **Revisión**: es resultado de revisar una base de conocimiento por una fórmula es una base de conocimiento que no necesariamente extiende ni es parte de la base de conocimiento original. En general, si la fórmula por la que se hace la revisión no es una falsedad lógica entonces esta fórmula es *consistentemente* inferida de la base de conocimiento resultante.

Sin dudas, el modelo AGM ha sido el trabajo más influyente en la teoría de cambio de creencias. Sin embargo, hay modelos equivalentes tales como: *safe contractions* [AM85], *epistemic entrenchment* [GM88], *sphere systems* [Gro88], y *kernel contractions* [Han94]. Supongamos que K representa el conocimiento actual y α representa una nueva pieza de información. Supongamos que una contracción de K por α es realizada. Siguiendo el modelo AGM, una *partial meet contraction* está basada en una selección entre los subconjuntos maximales de K que no implican α. Alternativamente, otra posibilidad es realizar una selección entre los subconjuntos minimales de K que implican α tal como en safe contraction [AM85]. Una variante más general del mismo enfoque fue introducido más tarde y es conocido como *kernel contraction* [Han94]. Como veremos más adelante, este enfoque propuesto por Hansson es muy natural para nuestro entorno de aplicación de bases de conocimiento finitas, y será la piedra basal del desarrollo de los operadores introducidos en el presente trabajo.

Además de las tres operaciones básicas mencionadas, a través de los años operaciones adicionales fueron desarrolladas en Revisión de Creencias para conseguir diferentes cometidos. Por ejemplo, cuando una base de conocimiento es inconsistente, la remoción (*esto es,* contracción) de suficientes sentencias de la misma puede llevar a un estado de consistencia. Esta operación adicional, denotada por $K!$ (donde K es una base de conocimiento), es conocida como *consolidación de conocimiento* [Han91a, Han01]. Esta operación es inherentemente diferente de la contracción y la revisión, ya que se enfoca en obtener conocimiento consistente a partir de una base de conocimiento (posiblemente) inconsistente, en lugar de revisar el conocimiento o remover alguna fórmula particular del mismo. Una manera natural de obtener la consolidación de conocimiento es utilizando la *contracción por falso* [Han91a], *esto es,* $K! = K \div \bot$, donde $\div$ representa un operador de

contracción. Cabe aclarar que este enfoque es válido solamente cuando trabajamos con bases de creencias (conjuntos de sentencias que no son cerrados bajo clausura lógica), ya que los conjuntos de creencias (conjuntos cerrados bajo clausura) inconsistentes equivalen al lenguaje.

1.3. Descripción del enfoque utilizado

Como fue explicado anteriormente, las inconsistencias e incoherencias que puedan aparecer en procesos automatizados tales como la integración de datos (data integration) y la adaptación de ontologías (ontology matching) pueden ser problemas graves en aplicaciones del mundo real; como los lenguajes ontológicos estándar adhieren a las semánticas clásicas de la Lógica de Primer Orden, las semánticas clásicas de inferencia fallan terriblemente en presencia de tales problemas. Por lo tanto, la definición de procesos capaces de resolver conflictos es una práctica de suma importancia en IA. En particular, en la presente tesis nos centramos en la formalización de procesos de resolución de conflictos dentro de conocimiento ontológico; en particular, se ataca el problema del manejo de las inconsistencias e incoherencias que puedan surgir en ontologías Datalog$^{\pm}$. Debido a las particularidades de los entornos ontológicos mencionadas anteriormente (*esto es,* la separación del conocimiento entre hechos y reglas de inferencia), los operadores que presentaremos en este trabajo involucran el manejo de ambos tipos de conflictos que surgen a partir de la integración y/o evolución de sistemas de información, *esto es,* tanto inconsistencias como incoherencias.

Como fue explicado, dos tipos de enfoques se pueden utilizar para el manejo de conflictos. En la presente tesis adherimos al primer enfoque presentado, *esto es,* tratar de modificar la información contenida en el conocimiento modelado por las ontologías de forma tal que mantengamos al mismo consistente (y, en entornos ontológicos, también coherente); esto es conocido en la literatura como *consolidación de conocimiento* [Han91a, Han01]. Con ese objetivo en vista proponemos formalismos generales que apuntan hacia la consolidación (*esto es,* a la resolución de cada conflicto de consistencia o coherencia) en ontologías Datalog$^{\pm}$. Esto es, los operadores propuestos en la tesis toman como entrada una ontología Datalog$^{\pm}$ (la cual es potencialmente inconsistente e incoherente) y retorna otra ontología Datalog$^{\pm}$ donde cada conflicto presente en la original fue subsanado. Siguiendo la tradición en la literatura, un requisito adicional para los procesos de consoli-

dación presentados es que los mismos alteren las ontologías tan poco como sea posible. El enfoque utilizado se basa en *Kernel Contraction* [Han93, Han94, Han97b, Han01], el cual se centra en la resolución de conflictos mínimos conocidos como kernels. De esta forma, el proceso de resolución de conflictos es definido como la identificación de tales conflictos mínimos y la posterior remoción de alguna fórmula de los mismos para de tal forma resolver cada uno de ellos. La ontología resultante del proceso de consolidación es aquella formada por todas las fórmulas de la ontología original que no hayan sido removidas por los operadores.

Para la consecución del comportamiento recientemente descrito es necesario primeramente definir y desarrollar conceptos previos que permitan la posterior formalización de los operadores de consolidación. En primer lugar, a pesar del gran auge en los últimos tiempos de los lenguajes ontológicos en general y Datalog$^{\pm}$ en particular, muy pocos conceptos de Representación de Conocimiento han sido trasladados a este último desde otros formalismos más tradicionales. En esta línea es especialmente notoria la falta de una definición formal del concepto de incoherencia en ontologías Datalog$^{\pm}$. Es por esto que otro de los objetivos en el presente trabajo es establecer formalmente tal concepto, tomando como punto de partida los esfuerzos realizados para otros formalismos como Lógicas Descriptivas (Description Logics - DLs), así como también profundizar en la relación entre los conceptos de inconsistencia e incoherencia, para determinar la influencia de los mismos (y su interacción) en los procesos de consolidación.

Finalmente, como se expresó anteriormente el proceso de consolidación se basa en la remoción de fórmulas en conjuntos conflictivos mínimos. En la presente tesis se formalizan dos operadores distintos, que corresponden con intuiciones diferentes. Por un lado se definen operadores locales, los que intuitivamente tratan a cada kernel por separado, eligiendo para remoción aquella fórmula que menos información aporta dentro del kernel. Por otro lado, otro tipo de operadores es definido, conocidos como operadores globales, los cuales aprovechan la interacción entre kernels (cuando la misma existe) para conseguir una consolidación mas eficiente en términos de mínima pérdida de información. Durante el trabajo se analizan ambos enfoques, identificando fortalezas y debilidades de los mismos que justifican la existencia de ambos. Además, para ambos tipos de operadores se presentan las propiedades esperadas para los mismos a través de *postulados*, así como también construcciones para tales operadores; estableciendo finalmente la relación existente entre las propiedades y las construcciones.

1.4. Contribuciones

Las principales contribuciones incluidas en esta disertación versan en diferentes direcciones. Primero, incluso cuando los conceptos de incoherencia e inconsistencia están relacionados, los mismos son problemas muy distintos en entornos de conocimiento ontológico donde hay una clara separación de los conocimientos intencionales y extensionales. Por lo tanto, ademas de definir y analizar ambas nociones en Datalog$^{\pm}$, en la presente tesis se ahonda en como las mismas se relacionan, y cómo tal relación hace que las mismas deban ser tratadas en combinación pero de forma separada. Luego, procedemos a definir las propiedades de operadores que trabajan de manera local a cada kernel, proveyendo a su vez una construcción posible para un operador que respete tales propiedades. Incluso cuando la operación de consolidación es una sola, tal operador puede verse como trabajando en fases, de forma tal que primero resuelve los problemas de incoherencia para luego subsanar las inconsistencias remanentes, antes de dar la ontología final resultante. Finalmente, un enfoque alternativo es propuesto que explota la interacción entre los distintos kernels en lugar de tratar a cada uno por separado. Tal enfoque hace posible que la pérdida de información sea mínima, teniendo como contrapartida una posible carga extra al no trabajar con conflictos mínimos.

Más específicamente, las contribuciones principales del presente trabajo son las siguientes:

1.4.1. Formalización del concepto de incoherencia para Datalog$^{\pm}$.

Incluso cuando es un concepto muy importante para la representación de conocimiento en entornos ontológicos, la noción de incoherencia en ontologías Datalog$^{\pm}$ no había sido hasta el momento formalizada. Por esto, en el presente trabajo proveemos una definición formal de la misma, adaptando para tal fin una noción similar utilizada para las Lógicas Descriptivas. La formalización esta basada en la identificación de conjuntos de reglas utilizadas para la inferencia de nuevo conocimiento tales que las reglas en el conjunto no pueden ser aplicadas en su totalidad sin caer en una violación de las restricciones que se han impuesto al conocimiento. Además, ahondamos en la relación de la incoherencia con la inconsistencia, enfocándonos especialmente en como tal relación afecta la calidad

de los procesos de consolidación de ontologías Datalog$^{\pm}$, particularmente la pérdida de información, al punto de ser necesario definir operadores que tratan ambos problemas pero de forma separada.

1.4.2. Establecimiento de las propiedades esperadas para procesos de consolidación de ontologías Datalog$^{\pm}$.

Brindamos conjunto de propiedades que se espera sean satisfechas por operadores de consolidación de ontologías Datalog$^{\pm}$, adaptando las mismas a los casos de operadores trabajando a nivel local o global según el caso. Tales propiedades son presentadas a través de postulados, considerando algunas intuiciones en enfoques tradicionales para la Revisión de Creencias los cuales son adaptados al entorno ontológico de Datalog$^{\pm}$ (pero que pueden ser adaptados para que se adecúen a otros lenguajes ontológicos). Tales postulados proveen una caracterización formal de los operadores de consolidación sin adentrarse en *cómo* el proceso de consolidación es efectivamente realizado, proveyendo de esta forma un marco de comparación formal de operadores de consolidación de ontologías Datalog$^{\pm}$, tanto los definidos en la presente tesis como cualquier otro que trabaje en tales entornos.

1.4.3. Definición de construcciones para operadores de consolidación locales y globales.

En la presente tesis introducimos construcciones nóveles completas tanto para operadores de consolidación que tratan kernels por separado como para aquellos que explotan la relación de los mismos en un enfoque global. Tales construcciones toman una ontología Datalog$^{\pm}$ (posiblemente) inconsistente e incoherente y devuelve como resultado un subconjunto consistente y coherente de la misma. Una característica destacable de los operadores es que los mismos involucran enfoques desdoblados, donde primero el operador considera conflictos de incoherencia y resuelve los problemas de consistencia en un paso subsecuente, lo cual ayuda a mejorar el resultado final obtenido.

1.5. Publicaciones

Seguidamente se mencionan los principales trabajos realizados en el marco del desarrollo de la tesis. Los aportes publicados en tales artículos forman parte sustancial del presente trabajo, encontrándose los mismos diseminados en distintos capítulos de la disertación.

- En el artículo *"On the Influence of Incoherence in Inconsistency-tolerant Semantics for Datalog$^{\pm}$"* [DMFS15a], aceptado en el workshop "Ontologies and Logic Programming for Query Answering (ONTOLP)" asociado con la 24th International Joint Conference on Artificial Intelligence (IJCAI-2015) se introduce la formalización del concepto de incoherencia en ontologías Datalog$^{\pm}$, así como su relación con el concepto de inconsistencia y cómo el mismo influye en el comportamiento de muchas semánticas clásicas de tolerancia a inconsistencia. Los resultados publicados en tal artículo pueden encontrarse en la presente disertación en el Capítulo 3.

- En el artículo *"Datalog$^{\pm}$ Ontology Consolidation"* [DMFS15b], bajo revisión en la revista *Journal of Artificial Intelligence Research (JAIR)* se encuentran algunos de los más significativos resultados obtenidos durante este trabajo; a saber, para tal trabajo hemos desarrollado un operador de consolidación de ontologías basado en un enfoque local, tanto a nivel propiedades como así también proveyendo una construcción para el operador. Los resultados publicados en tal artículo pueden encontrarse en la presente disertación en el Capítulo 5.

- En los trabajos *"Inconsistency Resolution and Global Conflicts"* [DMFS14a], publicado en *European Conference on Artificial Intelligence (ECAI'14)*, e *"Improving Inconsistency Resolution by Considering Global Conflicts"* [DMFS14b], publicado en *Scalable Uncertainty Management (SUM'14)*, hemos presentado un primer paso hacia los operadores de consolidación globales introducidos en la presente tesis. En tal trabajo se introducen ejemplos de como los operadores locales pueden en ocasiones quedar atascados en máximos locales. Para prevenir tal comportamiento en ese trabajo se formalizan operadores globales que toman la noción de *clusters* del área de Bases de Datos y lo explotan para la definición de procesos de resolución de conflictos que tengan en cuenta la relación entre diferentes kernels. Es decir, los operadores presentados en tales trabajos proveen los fundamentos para el desarrollo

de los operadores globales de consolidación de ontologías Datalog$^{\pm}$ introducidos en la presente disertación en el Capítulo 6.

1.6. Organización de la tesis

La organización de la tesis es como sigue:

- **Capítulo 2.** En este capítulo se presenta en forma resumida los conceptos más importantes de la representación de conocimiento y razonamiento (Knowledge Representation and Reasoning - KR&R). Además, se introduce una recapitulación de los más importantes formalismos tradicionales de KR&R (tales como Lógica Proposicional y Lógica de Primer Orden), así como también en entornos ontológicos (presentando lenguajes como Description Logics y Programación Lógica).

- **Capítulo 3.** En este capítulo ahondamos en la presentación del formalismo de KR&R en el que se desarrollan las ontologías atacadas por los operadores de consolidación desarrollados en el presente trabajo. Primeramente se presentan los elementos básicos de Datalog$^{\pm}$ que permiten la representación de conocimiento, para luego introducir conceptos cruciales para la consolidación de ontologías como ser el de consistencia. Además, se presenta la formalización del concepto de incoherencia, así como un análisis de su relación con la inconsistencia.

- **Capítulo 4.** Aquí se presenta una recapitulación de los desarrollos más importantes dentro de la teoría de Revisión de Creencias, los cuales han sentado las bases para el desarrollo de los operadores presentados en la presente disertación. Es particularmente interesante la distinción entre las distintas operaciones posibles y como las mismas son en ocasiones interdefinibles entre ellas. Debido al objetivo de la presente tesis, es especialmente importante la relación entre las operaciones de consolidación y las operaciones de contracción, que hacen posible el desarrollo del enfoque presentado en este trabajo donde el proceso de consolidación es definido en base a la contracción de una falsedad lógica $\bot$.

- **Capítulo 5.** Este capítulo presenta el enfoque local de resolución de inconsistencias e incoherencias basado en Kernel Contraction. En el mismo comenzamos presentando el modelo epistémico utilizado en la definición de procesos de consolidación,

para luego introducir el conjunto de propiedades esperadas para tales operaciones mediante postulados. Una vez que tales propiedades son introducidas proseguimos con la definición de construcciones para operadores que satisfagan las mismas. Tales operadores trabajan sobre conjuntos mínimamente inconsistentes/incoherentes conocidos como kernels, y basan la resolución de conflictos en el uso de funciones de incisión que eligen formulas dentro de los kernels que serán removidas por el operador de consolidación. Finalmente, las propiedades y la construcción propuesta son relacionadas a través de un teorema de representación que establece que cualquier construcción que satisfaga las propiedades se corresponde necesariamente con la construcción propuesta en el presente trabajo, y un ejemplo completo del comportamiento del operador es presentado.

- **Capítulo 6.** En este capítulo comenzamos por mostrar como en determinadas ocasiones los operadores locales conllevan resultados indeseados, al ser considerados a la luz de la mínima pérdida de información. En particular, cuando existen kernels relacionados de alguna forma entonces la elección local en estos kernels relacionados no es siempre aconsejable. Para subsanar tal situación en este capítulo presentamos un postulado adicional que versa sobre la *optimalidad* de los operadores de consolidación, mostrando además un ejemplo que ilustra como los operadores locales no siempre satisfacen tal propiedad. Luego, se procede a introducir un enfoque alternativo que explota la relación entre los kernels mediante la definición de estructuras, denominadas *clusters*, que agrupan kernels relacionados. Se introducen nuevas funciones de incisión que trabajan sobre estas nuevas estructuras no-mínimas, y se muestra como la construcción propuesta, denominada *Cluster Contraction*, satisface el requerimiento de optimalidad propuesto a través de un nuevo teorema de representación. Finalmente se ilustra el comportamiento de estos operadores a través de un ejemplo completo.

- **Capítulo 7.** Distintas opciones de aplicación de los operadores presentados en la presente tesis son introducidas en este capítulo, como así también diferentes líneas de investigación que han surgido en el desarrollo de la misma. Dentro de las opciones presentadas una particularmente interesante es el uso de los operadores definidos en entornos de Bases de Datos Relacionales, lo que puede servir como un proceso de apoyo para la consecución de un objetivo más grande dentro de la línea de investigación en la cual se inserta el presente trabajo: la definición de procesos de

argumentación masiva sobre bases de datos federadas. Los procesos de consolidación aquí presentados pueden efectivamente servir de apoyo a tales sistemas, ya que la integración de diferentes bases de datos puede realizarse mediante una representación de las distintas bases de datos a través de ontologías Datalog$^{\pm}$ primeramente, para luego combinar todas las ontologías mediante unión conjuntista y finalmente aplicar operadores de consolidación en la unión para resolver los conflictos que pueda surgir. La ontología Datalog$^{\pm}$ final obtenida de la consolidación puede entonces verse como una vista unificada de todas las bases de datos involucradas en el sistema.

Otras aplicaciones de los operadores son discutidas, como ser la posibilidad de utilizar la consolidación como medio para definir operadores de revisión de ontologías en general, y operadores priorizados en particular. Tales operadores pueden ser de gran utilidad a la hora de diseñar, por ejemplo, procesos de evolución de ontologías. En particular, con tales operadores se podría apoyar la consecución de uno de los objetivos más importantes de la evolución: la modificación de la ontología para reflejar cambios en el mundo que la misma modela manteniendo las propiedades de consistencia y coherencia.

Finalmente, para culminar el capítulo procedemos a identificar distintas líneas de investigación que han surgido a partir del desarrollo presentado en la presente tesis, y que esperan ser atacadas a corto y mediano plazo.

- **Capítulo 8.** En este capítulo se ofrece primeramente una comparación de distintos enfoques clásicos de resolución de inconsistencias e incoherencias con los operadores presentados en esta tesis. A continuación, para finalizar la disertación se presenta un compendio de las conclusiones más importantes que se han obtenido en el desarrollo del presente trabajo.

Capítulo 2

Formalismos de representación de conocimiento

La Representación del Conocimiento y el Razonamiento (Knowledge Representation and Reasoning - KR&R) es un área de la inteligencia artificial cuyo objetivo fundamental es expresar conocimiento acerca de un dominio de forma tal que facilite la inferencia (sacar conclusiones, hacer explícita información implícita) a partir de dicho conocimiento, de forma tal que problemas complejos puedan ser resueltos. En general, esto involucra la utilización de un sistema de símbolos para representar un dominio del discurso (aquello de lo que se puede hablar), junto con funciones que permitan inferir (realizar un razonamiento formal) sobre los objetos. Esto es, la representación de conocimiento va de la mano con la definición de métodos automáticos, y prácticamente todo lenguaje de representación de conocimiento tiene un motor de razonamiento o inferencia como parte del sistema [HRWL84].

Tradicionalmente, los sistemas de representación de conocimiento versan sobre representaciones compuestas por objetos explícitos (*v.g.,* una clase que contiene a todos los objetos de tipo persona, o un individuo en concreto como ser Pedro), y de afirmaciones sobre ellos ("Pedro es una persona", o "todas las personas tiene un padre y una madre"). Representar el conocimiento en una forma explicita como esta permite a una computadora sacar conclusiones del conocimiento previamente almacenado (en el caso presentado el sistema podría inferir que "Pedro tiene un padre y una madre"). A través de los años diferentes lenguajes para la representación de conocimiento y razonamiento han sido desarrollados. En el presente capítulo recapitularemos algunos de los más im-

portantes desarrollos, comenzando por aquellos más tradicionales para luego enfocarnos en formalismos más recientes donde el conocimiento es expresado a través de reglas ontológicas, como el lenguaje en que se centrarán los desarrollos en la presente tesis. En particular, en el presente capítulo veremos cómo dos conceptos primordiales a la presente tesis han surgido en ciertos entornos ontológicos: incoherencia e inconsistencia.

2.1. Formalismos tradicionales de representación de conocimiento

Dentro de las distintas opciones disponibles para representar conocimiento las dos más difundidas son la Lógica Proposicional y la Lógica de Predicados, al punto que una gran parte de los desarrollos en la Teoría de Cambios se enfocan en bases de conocimiento expresadas en tales formalismos. A continuación introduciremos estos formalismos.

2.1.1. Lógica Proposicional

La lógica proposicional (a veces denominada lógica de orden cero) es la más antigua y simple de las formas de lógica. La misma es un sistema formal cuyos elementos más simples representan proposiciones, y cuyas constantes lógicas, llamados conectivos proposicionales, representan operaciones sobre proposiciones, capaces de formar otras proposiciones de mayor complejidad [EE01]. El razonamiento en la lógica proposicional es logrado a través de un mecanismo que primero evalúa sentencias simples y luego sentencias complejas, *esto es,* este mecanismo determina la veracidad de una sentencia compleja analizando los valores de veracidad asignados a las sentencias simples que la conforman. Hay tres elementos esenciales en los lenguajes lógicos: un alfabeto, una sintaxis, y una gramática. A continuación describiremos cada uno de estos elementos dentro de la lógica proposicional.

Alfabeto de la Lógica Proposicional

El alfabeto de un sistema formal es el conjunto de símbolos que pertenecen al lenguaje del sistema. Este determina cuales son los símbolos lógicos que permitirán traducir el

lenguaje coloquial en el lenguaje simbólico. Para el caso de la lógica proposicional el mismo consiste en:

- Una cantidad finita pero arbitrariamente grande de variables proposicionales. En general se las toma del alfabeto latino, utilizando *v.g.,* p, q, r, s, t, etc., y se usan subíndices cuando es necesario o conveniente. Una variable proposicional representa una proposición, y a su vez cada proposición se dice que es un caso de sustitución de una variable proposicional. Así, por ejemplo, "La casa está embrujada" se simboliza mediante la letra p, y es un caso de sustitución para p.

- Un conjunto de operadores lógicos representados con los signos $\sim$ (negación), $\wedge$ (conjunción), $\vee$ (disyunción), $\Rightarrow$ (condicional), $\Leftrightarrow$ (bicondicional), u otros, según las convenciones adoptadas por cada autor, los cuales sirven como nexo o conexiones entre proposiciones para construir proposiciones más complejas. A continuación denominaremos a las proposiciones como proposiciones simples, y a las proposiciones complejas (las cuales contienen dos o más proposiciones simples unidas mediante el uso de algún conectivo) como proposiciones compuestas.

- Dos signos de puntuación, paréntesis izquierdo y derecho. Su única función es la de eliminar la ambigüedad en ciertas expresiones compuestas.

Sintaxis de la Lógica Proposicional

La *sintaxis* definida en el lenguaje lógico establece la forma correcta de escribir las proposiciones en el lenguaje lógico proposicional. Para emplear correctamente los elementos descriptos anteriormente es preciso crear reglas de escritura, con el objetivo de crear "fórmulas bien formadas". Una fórmula bien formada se define de acuerdo a las siguientes reglas:

1) Las variables proposicionales p, q, r, s, t, etc., son fórmulas bien formadas.

2) Si A y B són fórmulas bien formadas, entonces su conexión a través de un conectivo lógico es también una fórmula bien formada. Es decir, son fórmulas bien formadas las siguientes: $\sim A$, $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $A \Leftrightarrow B$.

3) Sólo son fórmulas bien formadas aquellas que pueden ser generadas en un número finito de pasos respetando las condiciones 1) y 2).

Para producir una correcta interpretación de la relación entre variables proposicionales y conectivos, cuando hay más de un conectivo, se definen las siguientes reglas: (a) un conectivo afecta a las letras proposicionales inmediatas o a algún conjunto de letras y símbolos inmediatos a ellos entre paréntesis; y (b) para evitar el exceso de paréntesis se define una jerarquía entre conectivos los cuales se muestran en la siguiente tabla:

Nivel de precedencia	Conectores
1	$\sim$ (negación),
2	$\wedge$ (conjunción),
3	$\vee$ (disyunción),
4	$\Rightarrow$ (condicional),
5	$\Leftrightarrow$ (bicondicional)

Semántica de la Lógica Proposicional

Une vez definida la parte *sintáctica* del lenguaje, estamos en condiciones de introducirnos en la *semántica* de un lenguaje lógico proposicional. Intuitivamente podemos afirmar que la semántica nos informa el significado de las proposiciones en el mundo real. En este sentido, las conectivas generan un significado dentro de las proposiciones compuestas, a partir de las proposiciones componentes que conectan.

En la lógica proposicional, las conectivas lógicas se tratan como funciones de verdad. Es decir, como funciones que toman conjuntos de valores de verdad y devuelven valores de verdad. A cada proposición simple se le asigna uno de dos posibles valores de verdad: V (Verdadero) o F (Falso). Esto quiere decir que si hay n proposiciones simples dentro de una fórmula proposicional, la cantidad de posibles asignaciones de valores de verdad a las proposiciones simples que componen la fórmula proposicional es 2^n. Por ejemplo, la conectiva lógica "$\sim$" es una función que si toma el valor de verdad V, devuelve el valor de verdad F, y si toma el valor de verdad F, devuelve V. Por lo tanto, si se aplica la función "$\sim$" a una letra que represente una proposición *Falsa*, el resultado será *Verdadero*. Si es falso que "está lloviendo" (p), entonces será verdadero que "no está lloviendo" ($\sim p$).

Cada conectiva lógica se distingue de las otras por los valores de verdad que devuelve frente a las distintas combinaciones de valores de verdad que puede recibir. Esto quiere decir que el significado de cada conectiva lógica puede ilustrarse mediante una tabla, conocida como *tabla de verdad*, que despliegue los valores de verdad que la función devuelve frente a todas las combinaciones posibles de valores de verdad que puede recibir. En la

Figura 2.1 presentamos la tabla de verdad asociada a cada una de las operaciones definidas en la lógica proposicional.

ϕ	$\neg\phi$
V	F
F	V

ϕ	ψ	$\phi \wedge \psi$
V	V	V
V	F	F
F	V	F
F	F	F

ϕ	ψ	$\phi \vee \psi$
V	V	V
V	F	V
F	V	V
F	F	F

ϕ	ψ	$\phi \rightarrow \psi$
V	V	V
V	F	F
F	V	V
F	F	V

ϕ	ψ	$\phi \leftrightarrow \psi$
V	V	V
V	F	F
F	V	F
F	F	V

Figura 2.1: Tablas de verdad de los conectivos lógicos.

Se denomina interpretación de una fórmula a la asignación de un valor de verdad para cada una de sus fórmulas componentes básicas, es decir, una interpretación puede verse como una línea de la tabla de verdad que tiene asociada la fórmula.

La evaluación semántica de las fórmulas lógicas supone determinar el valor de verdad de la fórmula para todas las posibles valuaciones, es decir, se debe establecer el valor de verdad para todas las posibles combinaciones existentes, de esta manera, es posible crear la tabla de verdad asociada a la fórmula lógica en cuestión. Normalmente una fórmula, por la definición inductiva en su construcción, contiene subfórmulas, es decir, está formada por otras fórmulas mas simples. Por lo tanto, se debe establecer el valor de verdad de todas las subfórmulas en forma gradual, hasta finalmente obtener la evaluación de la fórmula completa. Las fórmulas lógicas pueden clasificarse en tres categorías, en donde dicha categorización depende de las valuaciones finales obtenidas para la fórmula en cuestión, las cuales son: fórmula válida o tautología (para cualquier valuación el valor de verdad de la fórmula es V), fórmula satisfascible o contingencia (existe al menos una valuación en donde la fórmula es V) y fórmula insatisfascible o contradicción (para cualquier valuación el valor de verdad de la fórmula es F).

2.1.2. Lógica de Predicados

La lógica proposicional trata con sistemas lógicos que carecen de cuantificadores, o variables interpretables como entidades. Para agregar mayor expresividad al lenguaje, permitiendo así representar enunciados coloquiales más complejos, surge la lógica de predicados. Este tipo de lenguaje lógico tiene el poder expresivo suficiente para definir a prácticamente todas las matemáticas.

Alfabeto de la Lógica de Predicados

El *alfabeto* del lenguaje lógico de predicados determina cuales son los símbolos lógicos que se utilizarán para traducir el lenguaje coloquial en el lenguaje simbólico, y consiste en:

- Un conjunto numerable de símbolos de variables $\mathcal{V} = \{X, Y, ...\}$, un conjunto de símbolos de constante $\mathcal{C} = \{a, b, c, ...\}$, un conjunto de símbolos de función $\mathcal{F} = \{f, g, h, ...\}$, y un conjunto de símbolos de predicado (o conjunto de símbolos de relación) $\mathcal{P} = \{p, q, e, ...\}$. Cada símbolo de función f (o predicado p) tiene asociado un número natural, denominado aridad, que indica el número de argumentos sobre los que se puede aplicar f (o p). Si la aridad de un símbolo f (o p) es n, se suele hacer explícito este hecho escribiendo f^n (o p^n). Es habitual considerar los símbolos de constante como símbolos de función de aridad cero.
- Un conjunto de operadores lógicos representados con los signos $\{\sim, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$, y los cuantificadores $\{\forall, \exists\}$.
- Un conjunto de signos de puntuación {‘(’,‘)’,‘,’}.

En virtud de las leyes de equivalencia lógica, no es estrictamente necesario el uso de las conectivas lógicas habituales $\{\wedge, \vee, \Leftrightarrow\}$, así como tampoco el cuantificador existencial $\exists$, dado que cualquier fórmula que las contiene puede transformarse en otra equivalente que no las usa. Se debe notar que el conjunto de puntuación se incluye para facilitar la legibilidad de las fórmulas; sin embargo, muchos autores no los consideran parte del lenguaje.

Sintaxis de la Lógica de Predicados

La *sintaxis* definida en el lenguaje lógico establece la forma correcta de escribir las proposiciones en el lenguaje lógico de predicados. En la lógica de predicados se denomina términos y fórmulas a las cadenas de símbolos, formados de acuerdo a las reglas inductivas especificadas a continuación, que constituyen construcciones bien formadas del lenguaje.

Reglas sintácticas para los términos y fórmulas:

- Si $t \in (\mathcal{V} \cup \mathcal{C})$ entonces t es un término; notaremos al conjunto de términos como $\mathcal{T}$.

- Si $t_1, t_2, ..., t_n$ son términos, y f es un símbolo de función entonces $f(t_1, t_2, ..., t_n)$ es un término.
- Si $t_1, t_2, ..., t_n$ son términos y p es un símbolo de relación entonces $p(t_1, t_2, ..., t_n)$ es una fórmula atómica.
- Toda fórmula atómica es una fórmula bien formada.
- Si A y B son fórmulas bien formadas, también lo son: $\sim A$, $\sim B$, $A \Rightarrow B$.
- Si A es una fórmula bien formada y $X \in \mathcal{V}$ entonces, $(\forall X)A$ es una fórmula bien formada. La variable X se denomina variable cuantificada.

Un término fijo (ground) es un término sin variables. Una fórmula fija (ground) es una fórmula bien formada que no contiene variables.

Dentro de la lógica de predicados, los cuantificadores tienen un determinado alcance que afecta a las variables de las fórmulas lógicas. Si un cuantificador no va seguido por el signo de puntuación paréntesis, su alcance llega hasta la(s) variable(s) correspondiente(s) a la aparición de la primera letra del predicado cuantificada comenzando desde la derecha. Si el cuantificador va seguido del signo de puntuación paréntesis, el alcance del cuantificador llega a toda la expresión encerrada. Se llaman variables ligadas a aquellas que caen bajo el alcance de un cuantificador, de lo contrario se denominan variables libres. Cuando una fórmula contiene todas sus variables ligadas por algún cuantificador, se denomina fórmula cerrada.

Semántica de la Lógica de Predicados

Así como sucede en la lógica proposicional, la semántica de la lógica de predicados nos permite conocer el valor de verdad de una determinada fórmula de la lógica de predicados. El significado de las fórmulas de un lenguaje de la lógica de predicados se determina en dos etapas: atribuyendo una interpretación a cada término utilizado precisando la entidad del universo modelado; y asignando un valor de verdad a los predicados según si la relación denotada se verifica o no en el universo. De esta manera, conociendo los valores de verdad de las fórmulas atómicas, se puede determinar el valor de verdad de una fórmula compuesta aplicando la semántica convencional de los símbolos lógicos.

Una interpretación nos da una manera de asignar significado a las construcciones del lenguaje de predicados, al cual denotaremos como $\mathcal{L}$. Una interpretación I de $\mathcal{L}$ es un par (D, J) que consiste en:

- Un conjunto no vacío D, el cual es el dominio de I.
- Una aplicación J que asigna:
  - A cada símbolo de constante, a de $\mathcal{L}$, un elemento distinguido del dominio D, $J(a) = a'$ donde $a' \in D$.
  - A cada símbolo de función n-ario f de $\mathcal{L}$ una función $J(f) = f'$, tal que $f' : D^n \to D$.
  - A cada símbolo de relación r n-ario de $\mathcal{L}$ una relación $J(r) = r'$, tal que $r' \subset D$; esto es un conjunto de n-tuplas de D, $r' = \{(d_1, ..., d_n) \mid d_i \in D\}$.

Sólo es posible hablar de verdad y falsedad en el contexto de una interpretación. Para ello, si la fórmula en cuestión tiene variables libres (es decir, no es cerrada) es necesario realizar un paso previo: asignar valores a las ocurrencias de las variables (libres) en la fórmula.

Valoraciones: Una valoración $\vartheta : \mathcal{T} \to D$ en I es una aplicación que asigna a cada término de $\mathcal{L}$ un elemento del dominio de la interpretación D y que se define inductivamente mediante las siguientes reglas:

$$\vartheta(t) = \begin{cases} v(X) & si\ t \in \mathcal{T} \wedge t \equiv X; \\ J(a) & si\ t \in \mathcal{C} \wedge t \equiv a; \\ J(f)(\vartheta(t_1), ..., \vartheta(t_n)) & si\ f \in \mathcal{F} \wedge (t_1 \in \mathcal{T} \wedge ... \wedge t_n \in \mathcal{T}) \wedge t \equiv f(t_1, ..., t_n); \end{cases}$$

donde v es una aplicación que asigna a cada variable de $\mathcal{L}$ un elemento del dominio de interpretación D.

Satisfacibilidad: Sea una interpretación $I = (D, J)$. Sea A una fórmula bien formada de $\mathcal{L}$. Decimos que la valoración ϑ en I satisface la fórmula bien formada A si y solo si se cumple que:

- Si $A \equiv r(t_1...t_n)$ entonces $r'(\vartheta(t_1)...\vartheta(t_n))$ es verdadero, donde $r' = J(r)$ es una relación en D.
- Si A es de la forma:
  - $\sim B$ entonces ϑ no satisface B.

- $(B \wedge C)$ entonces ϑ satisface B y ϑ satisface C.
- $(B \vee C)$ entonces ϑ satisface B o ϑ satisface C.
- $(B \Rightarrow C)$ entonces ϑ satisface $\sim B$ o ϑ satisface C.
- $(B \Leftrightarrow C)$ entonces ϑ satisface B y C, o ϑ no satisface ni B ni C.

- Si $A \equiv (\forall X)B$, para toda valoración ϑ, ϑ satisface B.
- Si $A \equiv (\exists X)B$, para alguna valoración ϑ, ϑ satisface B.

Verdad: Dada una fórmula bien formada de $\mathcal{L}$, es posible determinar la veracidad de la misma de la siguiente manera:

- Una fórmula bien formada A es verdadera en la interpretación I si y solo si toda valoración ϑ en I satisface A.
- Una fórmula bien formada A es falsa en I si y solo si no existe valoración ϑ en I que satisfaga A.
- A es lógicamente válida si y solo si para toda interpretación I, A es verdadera en I.
- A es insatisfacible si y solo si para toda interpretación I, A es falsa en I.
- A es satisfacible si y solo si existe una interpretación I y una valoración en ella de las variables libres tal que ϑ satisface A en I.

En la lógica de predicados, las fórmulas que provienen de tautologías (respectivamente, contradicciones), por sustitución de las variables enunciativas de éstas últimas por fórmulas bien formadas de la lógica de predicados, se denominan tautologías (respectivamente, contradicciones) del lenguaje.

2.2. Representación de conocimiento ontológico

Si bien la Lógica Proposicional y la Lógica de Predicados han sido ampliamente difundidos y utilizados, en los últimos tiempos nuevos formalismos han surgido, enfocados principalmente en su utilización en entornos colaborativos masivos, donde no sólo la capacidad de representación de conocimiento es importante, sino también el manejo de grandes

cantidades de datos y la tolerancia a las inconsistencias que puedan surgir a partir del mantenimiento colaborativo de las bases de conocimiento. A continuación presentaremos los desarrollos más importantes, a saber lenguajes ontológicos como las Lógicas Descriptivas, y otros formalismos relacionados que permiten la representación de conocimiento de forma declarativa como la Programación Lógica.

2.2.1. Lógicas Descriptivas (Description Logics - DL)

Las lógicas descriptivas (Description Logics - DL) son un conjunto de lenguajes de representación, los cuales pueden utilizados tanto para representar conocimiento acerca de individuos particular como así también de conceptos y relaciones entre éstos; de una forma estructurada y formalmente bien comprendida. El nombre lógica descriptiva se refiere por un lado a descripciones de conceptos usadas para describir un dominio y, por el otro, a la semántica que establece una equivalencia entre las fórmulas de lógicas de descripción y expresiones en lógica de predicados. Las lógicas descriptivas hoy en día se han convertido en una piedra fundamental de la web semántica para su uso en el diseño de ontologías.

Elementos del lenguaje

Hay diferentes componentes que forman parte del lenguaje utilizado para representar conocimiento en DL. Como es de esperarse, estos componentes versan o bien acerca de individuos, o bien acerca de información de carácter mas "general" como son los conceptos o roles entre éstos. Los elementos centrales del alfabeto del lenguaje de las lógicas descriptivas son:

- Nombres de concepto (concept name): asignan un nombre a un grupo de objetos.
- Nombres de rol (role name): asigna un nombre a una relación entre objetos.
- Nombres de individuos (u objetos): los individuos son instancias de los conceptos y también se pueden relacionar por medio de un rol.
- Constructores (constructor): relaciona nombres de conceptos y nombres de roles, y también crea conceptos complejos a partir de los atómicos (complex concepts).

- Definiciones de conceptos complejos: usa símbolos ($\doteq$, $\equiv$, $\sqsubseteq$) para declarar conjunto de igualdades, conjuntos de equivalencias y conjuntos de inclusiones.

Sintaxis

La sintaxis de un lenguaje de la familia de lógicas descriptivas nos informa acerca de los constructores que es posible utilizar en la definición de conceptos complejos (lo que obviamente implica que haya diferencias entre la expresividad de las distintas lógicas descriptivas). Algunos constructores están relacionados con constructores lógicos en la lógica de primer orden, como ser por ejemplo la disyunción y la conjunción de conceptos que se corresponden con los conectivos $\vee$ y $\wedge$, mientras que otros no tienen correspondencia, como son las restricciones en roles como la transitividad. La notación de los distintos elementos en DL es como se ve en la Tabla 2.2 en página 28.

Como dijimos antes las lógicas descriptivas son una familia de lenguajes que comparten características pero que como es de esperarse tienen sus diferencias. La tabla de constructores presentada recientemente es una tabla general, y distintas lógicas pueden definirse mediante la prohibición o no del uso de estos, afectando de esta forma la expresividad y complejidad de las mismas. Existe una convención (informal) acerca de la nomenclatura de las lógicas descriptivas que trata de codificar en el nombre de las mismas su expresividad, partiendo de tres logicas base y agregando modificadores de acuerdo a los constructores permitidos. Las tres lógicas básicas son:

- $\mathcal{AL}$: Lenguaje basado en atributos. Permite:
  - Negación atómica (negación de conceptos que no aparezcan en el lado izquierdo de axiomas),
  - intersección de conceptos,
  - restricciones universales,
  - y cuantificaciones existenciales limitadas.
- $\mathcal{FL}$: Lenguaje de descripción basado en marcos (frames). Permite:
  - Intersección de conceptos,
  - restricciones universales,

Notación	**Descripción**	**Ejemplo/Significado**
$\top$	Concepto especial donde todo individuo es una instancia del mismo	$\top$
$\bot$	Concepto vacío	$\bot$
$\sqcap$	Conjunción o intersección de conceptos	$Persona \sqcap Mujer$ ($Persona$ y $Mujer$)
$\sqcup$	Disyunción o unión de conceptos	$Hombre \sqcup Mujer$ ($Hombre$ o $Mujer$)
$\neg$	Negación de concepto	$\neg Tenista$ (no es $Tenista$)
$\forall$	Restricción universal	$\forall esHumano.Persona$ (toda $Persona$ cumple el rol $esHumano$)
$\exists$	Restricción existencial	$\exists tieneHijo.Persona$ (existe una $Persona$ que $tieneHijo$)
$\sqsubseteq$	Inclusión de conceptos	$Padre \sqsubseteq Persona$ (todos los $Padre$ son $Persona$)
$\equiv$	Equivalencia de conceptos	$Mujer \equiv Persona \sqcap Hembra$ Una $Mujer$ es una $Persona$ que es también $Hembra$
$\doteq$	Definición de conceptos	$Persona \doteq Humano$ ($Persona$ es definido como igual a $Humano$)
:	Aserción de concepto	$pedro : Persona$ ($pedro$ es $Persona$)
:	Aserción de rol	$(pedro, maria) : tieneHijo$ ($maria$ es la hija de $pedro$)

Figura 2.2: Notación de elementos en Description Logics

  - cuantificaciones existenciales limitadas,
  - y restricciones en roles.

- $\mathcal{EL}$. Permite:
  - Intersección de conceptos,
  - y restricciones existenciales.

Sobre estas lógicas podemos aplicar el siguiente conjunto de extensiones, de acuerdo a los constructores permitidos por la lógica que queremos denominar:

- $\mathcal{F}$: Propiedades funcionales.
- $\mathcal{E}$: Restricciones existenciales completas (aquellas que no sólo utilizan $\top$).
- $\mathcal{U}$: Unión de conceptos.
- $\mathcal{C}$: Negación de conceptos complejos.
- $\mathcal{H}$: Jerarquía de roles (subPropertyOf)
- $\mathcal{R}$: Axiomas de inclusión de roles complejos limitados: reflexividad e irreflexividad, roles disjuntos.
- $\mathcal{O}$: Nominales.
- $\mathcal{I}$: Propiedades de inversión, como roles inversos.
- $\mathcal{N}$: Restricciones de cardinalidad.
- $\mathcal{Q}$: Restricciones de cardinalidad calificadas (aquellas que no sólo usan $\top$).
- $^{(\mathcal{D})}$: Uso de tipos de datos y sus propiedades.

Sin embargo, si bien el uso de tal notación está bastante difundido y respetado, también podemos encontrar excepciones en algunas lógicas que no la siguen. Ejemplos importantes de estas desviaciones respecto de la notación usual son $\mathcal{S}$ que se utiliza como abreviación de $\mathcal{ALC}$ y $\mathcal{EL}^{++}$ el cual es un alias para $\mathcal{ELRO}$.

Entre las lógicas descriptivas más utilizadas y difundidas tenemos las siguientes:

- $\mathcal{ALC}$: es una de las lógicas más importantes, y la cual se utiliza de benchmark a la hora de comparar expresividad entre lógicas. Como su nombre lo indica, $\mathcal{ALC}$ es simplemente $\mathcal{AL}$ con el agregado de que las negaciones pueden darse en cualquier concepto y no sólo en los atómicos.
- $\mathcal{FL}^{-}$: un sublenguaje de $\mathcal{FL}$ obtenido a través de la prohibición de usar restricciones en roles.
- $\mathcal{SHIQ}$: surge de aumentar $\mathcal{ALC}$ con roles transitivos (jerarquía de roles - $\mathcal{Q}$), roles inversos ($\mathcal{I}$) y restricciones cardinales extendidas ($\mathcal{Q}$).
- $\mathcal{EL}^{++}$.

Relación de DL con OWL

Además de las recientemente mencionadas, hay otras DLs que son importantes porque se corresponden con lenguajes de otra familia de lenguajes ontológicos interesantes para la Web Semántica: OWL (Web Ontology Language). OWL es otra familia de lenguajes para la definición de ontologías, caracterizado por tener, como en el caso de DLs, una clara formalización de su semántica. Sin embargo, si bien ciertas terminologías cambian entre DLs y OWLs (*v.g.,* un concepto en la jerga de DL se refiere a una clase en OWL, mientras que un rol en la jerga de DL es una propiedad en OWL), la realidad es que como la definición de muchos OWLs esta basada en diferentes DLs la expresividad entre los mismos se corresponde. Los ejemplos más importantes de esto son:

Lenguaje OWL	DL correspondiente
OWL DL	$\mathcal{SHOIN}^{(\mathcal{D})}$
OWL Lite	$\mathcal{SHIF}$
OWL 2	$\mathcal{SHROIQ}$

En la presente tesis no ahondaremos en los lenguajes OWL y nos enfocaremos principalmente en DLs; la razón de esto es que no sólo su expresividad está relacionada sino que además, como veremos más adelante en la tesis cuando discutamos trabajos relacionados a la misma, la gran mayoría de los esfuerzos realizados para el manejo de inconsistencia en entornos ontológicos tienen a DLs como ámbito de aplicación. El lector interesado puede dirigirse a [Hor05a, Hor05b] para un tratamiento más profundo tanto de los distintos OWLs en sí mismos como también su relación con las distintas DLs.

Bases de conocimiento en DL

Una base de conocimiento en DL está formada por dos componentes, el TBox (Terminological Box) y el ABox (Assertional Box), donde uno de ellos se enfoca en la descripción de elementos terminológicos (TBox) mientras que el otro se encarga de expresar conocimiento particular acerca de individuos (ABox) por lo que es a veces denominado *descripción del mundo (world description)*.

El TBox contempla toda la terminología, es decir, el vocabulario de un dominio de aplicación en función de:

- Conceptos: denotan clases o conjunto de individuos.
- Roles: denotan relaciones binarias entre los individuos.
- Un conjunto de descripciones complejas.

Las TBox son definidas como un conjunto finito de Inclusiones de Conceptos Generales (General Concept Inclusion - CGI) de la forma $C \sqsubseteq D$.

El ABox contiene aserciones acerca de individuos nombrados en términos de vocabulario. Esto es, una ABox es un conjunto finito de axiomas de aserción, que puede ser tanto una aserción de concepto como una de rol.

Finalmente, una base de conocimiento en DL es un par $KB = (\mathcal{T}, \mathcal{A})$ donde $\mathcal{T}$ es una TBox y $\mathcal{A}$ es una ABox. Una base de conocimiento en DL es equivalente a un conjunto de axiomas de la lógica de predicados, y por tanto se puede definir un cálculo o sistema de inferencia que permite derivar conocimiento implícito a partir del explícito de la base de conocimiento.

Semántica

La semántica de una lógica descriptiva es definida mediante la interpretación de conceptos como conjuntos de individuos (aquellos individuos que corresponden al concepto) y la interpretación de roles como pares ordenados de individuos (aquellos relacionados a través del rol). Tales individuos son asumidos para un dominio dado. A su vez, la semántica de conceptos y roles no atómicos es definida en términos de aquellos que sí son atómicos, mediante recursiones. Para hacer la presentación formal del aspecto semántico

de las lógicas descriptivas utilizaremos como base la descripción de $\mathcal{ALC}$, siguiendo las definiciones de Baader *et al.* [Baa03].

Una *signatura* es una tripla ordenada (N_C, N_R, N_O) donde N_C es el conjunto de nombres de conceptos, N_R es el conjunto de nombre de roles y N_O es el conjunto de nombres de objetos o individuos. Intuitivamente, la signatura es el universo de aquellas cosas sobre las que podemos expresar conocimiento. Dada una signatura, llamamos *interpretación terminológica* sobre la signatura al par $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ donde

- $\Delta^{\mathcal{I}}$ es un conjunto no vacío denominado el *dominio*, y
- $\cdot^{\mathcal{I}}$ es una función de interpretación que mapea
  - cada individuo $a \in N_O$ a un elemento $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
  - cada concepto $c \in N_C$ a un subconjunto de $\Delta^{\mathcal{I}}$
  - cada nombre de rol $r \in N_R$ a un subconjunto de $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

Este mapeo es definido de la siguiente forma:

- $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
- $\bot^{\mathcal{I}} = \emptyset$
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$ (la unión se trata como disyunción)
- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$ (la intersección se trata como conjunción)
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ (el complemento significa negación)
- $(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} |$ todo $y, (x, y) \in R^{\mathcal{I}}$ implica $y \in C^{\mathcal{I}}\}$
- $(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} |$ existe $y, (x, y) \in R^{\mathcal{I}}$ y $y \in C^{\mathcal{I}}\}$

La importancia de las interpretaciones es que nos dan una forma de saber si un mapeo particular (es decir, una interpretación) es modelo de un concepto, es decir, si el concepto es válido para una ontología DL, y es por lo tanto una consecuencia de la misma (bajo dicha interpretación). Mas formalmente, definimos a la relación de modelado $\models$ como aquella donde:

- Para la TBox:
  - $\mathcal{I} \models C \sqsubseteq D$ si y sólo si $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
  - $\mathcal{I} \models \mathcal{T}$ si y sólo si $\mathcal{I} \models \Phi$ para todo $\Phi \in \mathcal{T}$.
- Para la ABox:
- $\mathcal{I} \models a : C$ si y sólo si $a^{\mathcal{I}} \in C^{\mathcal{I}}$
- $\mathcal{I} \models (a, b) : R$ si y sólo si $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
- $\mathcal{I} \models \mathcal{A}$ si y sólo si $\mathcal{I} \models \phi$ para todo $\phi \in \mathcal{A}$.

Finalmente, con tales definiciones podemos establecer cuando una interpretación modela a una base de conocimiento en Description Logics. Básicamente, esto sucede cuando cada concepto y rol en las Tbox y ABox en la base de conocimiento tiene una contrapartida en la interpretación, *esto es,* para la base de conocimientos $KB = (\mathcal{T}, \mathcal{A})$ tenemos que $\mathcal{I} \models KB$ si y sólo si vale que $\mathcal{I} \models \mathcal{T}$ y que $\mathcal{I} \models \mathcal{A}$. A su vez, para un axioma cualquiera ϕ decimos que KB infiere a ϕ cuando para cada interpretación $\mathcal{I}$ tal que $\mathcal{I} \models KB$ tenemos que $\mathcal{I}$ satisface a ϕ.

Inconsistencia e incoherencia en Description Logics

Como hemos explicado en el Capítulo 1 de la presente disertación, en entornos ontológicos como los que sirven de ámbito de aplicación para operadores de consolidación hay dos tipos de conflictos que deben ser atacados. Si bien en el presente trabajo nos enfocamos en un formalismo de representación de conocimiento diferente a las Lógicas Descriptivas, esto es Datalog$^{\pm}$, es innegable la gran influencia de las mismas en el desarrollo de tales conceptos. Por lo tanto, a continuación recapitularemos brevemente tales conceptos dentro del ámbito de DL, lo que servirá para sentar las bases del tratamiento que daremos a tales conceptos en Datalog$^{\pm}$ en el Capítulo 3 (un tratamiento más profundo de ambos conceptos en DL pueden encontrarse en [Baa03] y [FHP^{+}06]).

Primeramente, como es esperable definimos a una base de conocimiento KB en DL como consistente si la misma es tal que tiene una interpretación posible. Es decir, KB es consistente si y sólo si existe $\mathcal{I}$ tal que $\mathcal{I} \models KB$. Por oposición, llamamos inconsistente a la base de conocimiento tal que no existe interpretación que la modele.

Para hablar de incoherencia primeramente se debe definir cuando un concepto en la TBox de una ontología DL es *satisfacible*. Intuitivamente, un concepto es satisfacible si el mismo *tiene sentido*, no es contradictorio; es decir, existe una interpretación donde los axiomas de la TBox son satisfechos y el concepto en cuestión no implica un conjunto vacío (*esto es,* hay al menos un individuo que corresponde al concepto). Si esta situación no es posible decimos que el concepto es insatisfacible. En base a la distinción entre conceptos satisfacibles e insatisfacibles podemos definir incoherencia para una ontología DL: decimos que KB es incoherente si tiene al menos un concepto insatisfacible en su TBox [FHP$^+$06].

Como veremos más adelante en el Capítulo 3, la caracterización de incoherencia a través de la satisfacibilidad o no de los conceptos dentro de una ontología sentó las bases para el estudio propuesto en la presente tesis de tal fenómeno en Datalog$^\pm$.

2.2.2. Programación Lógica (Logic Programming - LP)

Otro formalismo ampliamente extendido que permite la representación de conocimiento es el de Programación Lógica (Logic Programming - LP) [Llo87, BCM$^+$03]. Los orígenes de la programación en lógica pueden ser encontrados al principio de la década del 70, surgiendo principalmente a raíz de los esfuerzos realizados en la prueba automática de teoremas y en la inteligencia artificial. Fue introducida por Kowalsky y van Emden [Kow74, VEK76] y Colmeraner *et al.* [CKPR73], mientras que la primera implementación de un intérprete para ProLog, el lenguaje más difundido del paradigma, es debida a Roussel en el año 1975 [Rou75].

La principal noción en la programación lógica es el utilizar (un fragmento de) la lógica de primer orden para representar conocimiento, tanto explícito como aquel que puede ser obtenido a través de mecanismos de inferencia. Una característica crucial de la programación lógica como paradigma de programación es el descomponer un algoritmo en dos componentes disjuntas: la lógica y el control. La componente lógica establece *qué* es lo que debe resolverse, mientas que la componente de control establece *cómo* debe resolverse. La separación de estas dos componentes permite al programador concentrarse en la componente lógica de su algoritmo, dejando el control para que sea resulto por el sistema de programación en lógica. Es decir, en este paradigma el programador se centra mucho más en aspectos de representación de conocimiento que en otros paradigmas, ya que

los programas lógicos suelen enfocarse en descripciones del mundo en donde la aplicación es utilizada.

De esta forma, un programa lógico [Llo87] se constituye mediante un conjunto finito de axiomas que representa nuestro conocimiento y las hipótesis del problema a resolver, y una sentencia meta que representa el objetivo que queremos alcanzar. La ejecución de un programa lógico implica la demostración de la sentencia meta a partir de los axiomas dados.

Elementos del lenguaje

A continuación presentaremos los distintos elementos de que puede uno valerse a la hora de representar conocimiento en un programa lógico. Por razones de simplicidad sólo presentaremos brevemente los elementos más importantes del paradigma. El lector es referido a [Llo87, Das92] para una presentación más completa del mismo, y a [SS94, Bra01] para conocer más en detalle a ProLog.

Un concepto básico en la programación lógica es el de *término*. Un término es o una constante, o una variable, o una construcción de la forma $f(t_1, t_2, \ldots, t_n)$ donde f es un símbolo de functor de aridad n y $t_1, t_2, \ldots, t_n$ son a su vez recursivamente términos. A su vez, a la construcción $p(t_1, t_2, \ldots, t_n)$ tal que $t_1, t_2, \ldots, t_n$ son términos es denominada átomo (o predicado) para el nombre de predicado p. Finalmente, un literal es un átomo, o la negación de un átomo. Por convención, tanto los nombres de predicados como las constantes se denotan con letras minúsculas, mientras que para las variables se utilizan letras mayúsculas.

Ejemplo 2.1 *Como ejemplos de átomos podemos tener a* $padre(martin, david)$ *y* $q(X, Y, Z)$.

En base a los átomos, en programación lógica podemos definir a las *cláusulas de programa definido*, las cuales son reglas que conectan un conjunto de átomos con otro átomo que puede verse como una conclusión de los primeros. Una cláusula es una disyunción finita de literales, *esto es,* $l_1 \vee l_2 \vee \cdots \vee l_n$ donde cada l_i con $1 \leq i \leq n$ es un literal. Un caso particularmente interesante de cláusulas para la programación lógica son las *cláusulas Horn*, que son cláusulas con a lo sumo un literal positivo. Esta forma de cláusulas son básicas en la

programación lógica, ya que permiten representar el conocimiento utilizando implicaciones de la forma $l_1 \leftarrow l_2 \vee \cdots \vee l_n$, como una representación alternativa de $l_1 \vee \neg l_2 \vee \ldots \neg \vee l_n$. Esto es conocido también como una *cláusula de programa definido*. Es decir, una cláusula de programa definido es una construcción de la forma $A \leftarrow B_1, B_2, \ldots, B_n$ donde A es un átomo denominado *cabeza* o *consecuente*, y la conjunción de átomos $B_1, B_2, \ldots, B_n$ es el cuerpo de la regla. Intuitivamente, se puede entender a la semántica detrás de las reglas como "si para cada asignación de variable se da que $B_1, B_2, \ldots, B_n$ es verdadero, entonces A es verdadero" [Llo87].

Ejemplo 2.2 *Por ejemplo, la siguiente es una cláusula:* $abuelo(X, Z) \leftarrow padre(X, Y), padre(Y, Z)$.

Finalmente, un programa lógico es un conjunto finito de átomos y cláusulas. Los programas lógicos son, por ende, la forma de representar el conocimiento que se tiene acerca del mundo en la programación lógica.

Cabe aclarar que en la programación lógica clásica se utiliza la negación por falla, de la mano de una suposición de mundo cerrado (closed world assumption - CWA). Esto significa que si un literal no puede ser probado (es decir, su prueba *falla*) entonces se asume que el mismo no vale, lo cual tiene un impacto importante a la hora de realizar inferencias en base a un programa lógico, lo que se hace a través de la resolución de cláusulas SLD (Selective Linear Definite clause resolution), la cual es la regla de inferencia básica en programación lógica, al ser tanto sensata como completa en refutación para cláusulas Horn [Llo87]. A través de los años, distintas extensiones a la programación se han desarrollado para incluir otras características, como la negación clásica o fuerte. Cada una de estas extensiones ha incorporado ventajas, pero al mismo tiempo han traído otros problemas de índole conceptual y computacional. En 1990 Gelfond y Lifschitz presentan una extensión que posibilita el uso de negación clásica en programas lógicos [GL90]. A su vez, ese mismo trabajo es extendido en [KS91] para incluir la posibilidad de representar excepciones dentro de programas lógicos. Otra extensión importante es debido a Katsumi Inoue, quien en [Ino91] presenta un formalismo que permite la utilización de hipótesis default a los programas extendidos presentados en [GL90].

2.2.3. Programación Lógica Rebatible (Defeasible Logic Programming - DeLP)

A continuación introduciremos otro formalismo que posibilita la representación de conocimiento y su explotación en entornos colaborativos es la Programación en Lógica Rebatible (Defeasible Logic Programming - DeLP) [GS04]. Este formalismo permite modelar conocimiento que involucre información incompleta o potencialmente contradictoria. El mecanismo de inferencia sobre el cual está basado permite decidir entre conclusiones contradictorias y adaptarse fácilmente a entornos dinámicos. El hecho de representar y razonar sobre información tentativa (además de estricta) hacen de DeLP un atractivo sistema de representación de conocimiento y razonamiento en un gran número de dominios de aplicación [RGS07, GGS08, SMCS08, DFDG+13, TGGS14].

En DeLP un programa lógico rebatible está formado por un conjunto de *hechos*, *reglas estrictas*, *reglas rebatibles*, y *presuposiciones* permitiendo estas dos últimas la representación de información tentativa. Toda conclusión del programa estará sustentada por algún *argumento* que pueda construirse utilizando las reglas, los hechos del programa, y las presuposiciones. Cuando se utilicen reglas rebatibles o presuposiciones para decidir cuando una conclusión L puede aceptarse a partir de un programa lógico rebatible, se realizará un *análisis dialéctico*, construyendo argumentos a favor y en contra de la conclusión L.

Intuitivamente, tal proceso dialéctico involucra el hecho de que un argumento que sustenta una conclusión L puede ser atacado por otros argumentos *derrotadores* que lo contradigan y que pueden construirse a partir del programa. Dichos derrotadores podrán a su vez ser atacados, y así sucesivamente, generando una secuencia de argumentos llamada *línea de argumentación*. Cada línea de argumentación satisface ciertas propiedades, a fin de evitar que se generen argumentaciones falaces. El proceso completo considera para cada argumento todos sus posibles argumentos derrotadores, que en lugar de una única línea de argumentación, genera un conjunto de líneas que son representadas de una manera compacta por medio de un *árbol de dialéctica*. Este análisis dialéctico determinará si la conclusión en cuestión está *garantizada* o no, a partir del programa.

A continuación ahondaremos en la representación de conocimiento a través de un programa lógico rebatible y las nociones necesarias para realizar el análisis dialéctico que es la base para la definición del proceso de resolución de consultas en el formalismo.

Lenguaje de Representación de Conocimiento

El lenguaje que se utiliza en la programación lógica rebatible para representar el conocimiento del dominio de aplicación se define en términos de cuatro tipos de cláusulas:

- hechos, que son literales fijos, *esto es,* sin variables,
- reglas estrictas, denotadas "$L_0 \leftarrow L_1, \ldots, L_k$",
- reglas rebatibles, denotadas "$L_0 \prec L_1, \ldots, L_k$", y
- presuposiciones, denotadas "$L_0 \prec$"

  donde $L_0, L_1, L_2, \ldots, L_k$, con $k > 0$, son literales fijos.

Los hechos y reglas estrictas se emplean para representar conocimiento seguro, libre de excepciones, mientras que las reglas rebatibles son reglas de inferencia que se utilizan para representar información tentativa, la cual puede ser cuestionada.

La rebatibilidad es una característica destaca del formalismo, y que se manifiesta como de vital importancia en entornos colaborativos, que son siempre propensos a inconsistencia. Una regla rebatible, denotada *Cabeza* $\prec$ *Cuerpo* se lee como "razones para creer en *Cuerpo* proveen razones para creer en *Cabeza*", pero tal relación de consecuencia es más débil que la relación estricta, y por lo tanto puede ser objetada. Por otro lado, de la misma forma que el símbolo "$\prec$" es lo que distingue sintácticamente a una regla rebatible de una estricta, en DeLP, una presuposición será representada con un literal fijo seguido del símbolo "$\prec$", para diferenciarla sintácticamente de un hecho. La misma distinción respecto de la severidad de la afirmación entre reglas estrictas y rebatibles puede hacerse respecto de hechos y presuposiciones, de forma tal que una presuposición "$L \prec$" expresa que "existen razones para creer en L", con lo cual representa una afirmación mucho más débil que un hecho.

Un programa lógico rebatible es, entonces, un conjunto $\mathcal{P}$ de hechos, reglas estrictas, reglas rebatibles y presuposiciones. En un programa $\mathcal{P}$ identificaremos con Θ al conjunto de hechos, con Ω al conjunto de reglas estrictas, con Δ al conjunto de reglas rebatibles, y con Φ al conjunto de presuposiciones. Por conveniencia, en la literatura es también común denotar con Π al conjunto $\Theta \cup \Omega$, y con Δ^+ al conjunto $\Delta \cup \Phi$. Por lo tanto, en la literatura

es posible encontrar tanto programas lógicos rebatibles presentados como $\mathcal{P} = (\Pi, \Delta^+)$, así como también en otros casos $\mathcal{P} = (\Theta, \Omega, \Delta, \Phi)$.

Es importante destacar que si bien DeLP solo admite literales fijos en las reglas, siguiendo la misma política que Lifschitz en [Lif96], en la literatura de DeLP se suelte utilizar variables solamente como una forma de denotar *esquemas de reglas*. Para diferenciar las variables de los demás elementos del programa, al igual que en el caso de la programación lógica estas serán denotadas con una letra mayúscula inicial.

Un ejemplo de un programa DeLP es presentado a continuación.

Ejemplo 2.3 *El programa DeLP $\mathcal{P}$ expresa información sobre el mercado de valores.*

$$\mathcal{P} = \left\{ \begin{array}{l} comprar\_acciones(T) \prec buen\_precio(T) \\ \sim comprar\_acciones(T) \prec buen\_precio(T), empresa\_riesgosa(T) \\ \sim comprar\_acciones(T) \prec empresa\_riesgosa(T) \\ empresa\_riesgosa(T) \leftarrow huelga\_empleados(T) \\ huelga\_empleados(acme) \prec \\ buen\_precio(acme) \end{array} \right\}$$

Para representar la información tentativa, en este caso, se utilizaron cuatro reglas rebatibles y una presuposición. De la primer regla se puede concluir en comprar acciones de una empresa si hay razones para creer que están a buen precio. La segunda y tercer regla presentan razones en contra de comprar acciones: si las acciones están a buen precio pero son de una empresa riesgosa entonces hay razones tentativas para no comprar dichas acciones, a la misma conclusión se puede llegar si hay razón para creer que la empresa es una empresa en riesgo. La cuarta regla es una regla estricta que indica que si hay huelga de empleados en una empresa entonces tal empresa es riesgosa. Por último, debido a reportes que han llegado indicando de ciertos empleados no trabajando en la empresa se presupone que en la empresa "acme" hay huelga de empleados.
Finalmente, tenemos un hecho el cual afirma que las acciones de la empresa "acme" están a buen precio.

Argumentación Rebatible

Ya hemos visto cómo se representa el conocimiento en DeLP. En base a tales elementos se puede definir cómo derivar información en el formalismo. En particular, un concepto

importante es aquél de *derivación rebatible*, el cual identifica la forma en que literales pueden ser derivados utilizando las reglas de un programa lógico rebatible. Una derivación rebatible de L a partir de un programa $\mathcal{P}$, denotado $\mathcal{P} \mid\sim L$, consiste de una secuencia finita $L_1, L_2, \ldots, L_n = L$ de literales fijos tal que para cada i, $1 \leq i \leq n$, se cumple que:

- L_i es un hecho en $\mathcal{P}$, o
- es una presuposición en $\mathcal{P}$, o
- existe una regla R_i (estricta o rebatible) en $\mathcal{P}$ con cabeza L_i y cuerpo $B_1, B_2, \ldots, B_k$ y cada literal en el cuerpo B_j $(1 \leq j \leq k)$ de la regla es un elemento de la secuencia apareciendo antes que L_i.

Es importante notar que el hecho de que un literal L puede ser derivado de un programa *no* implica que L sea aceptado en el formalismo, puede existir en el programa información que contradiga a L, y entonces L puede no ser aceptado como una creencia válida del programa. Por ejemplo, a partir del programa $\mathcal{P}$ del Ejemplo 2.3 es posible obtener una derivación rebatible para el literal "*compra_acciones*(*acme*)", ya que existe la secuencia de literales: *buen_precio*(*acme*), *compra_acciones*(*acme*); que se obtiene utilizando el hecho "*buen_precio*(*acme*)", y la regla rebatible "*compra_acciones*(*acme*) $\prec$ *buen_precio*(*acme*)". Además, también es posible obtener una derivación para el literal "$\sim$*compra_acciones*(*acme*)", ya que existe la secuencia de literales: *huelga_empleados*(*acme*) $\prec$, *empresa_riesgosa*(*acme*), $\sim$*compra_acciones*(*acme*). Un caso especial de derivación es el de la *derivación estricta*, la que corresponde al caso donde todas las reglas utilizadas en la derivación son estrictas, *esto es,* L tiene una derivación estricta a partir de $\mathcal{P}$, denotado $\mathcal{P} \vdash L$, si todas las reglas de programa utilizadas para obtener la secuencia $L_1, L_2, \ldots, L_n = L$ son reglas estrictas.

Como las reglas de programa permiten utilizar literales negados en la cabeza, entonces es posible derivar literales complementarios; dos literales son contradictorios si son complementarios. Como las reglas rebatibles permiten derivar literales complementarios, el conjunto $\Pi \cup \Delta^+$ puede ser contradictorio. Un *conjunto de reglas es contradictorio* si y solo si es posible obtener derivaciones rebatibles para un literal L y su complemento $\overline{L}$, a partir de este conjunto. Sin embargo, desde el punto de vista de la representación de conocimiento en DeLP es importante aclarar que dado un programa lógico rebatible $\mathcal{P}= (\Pi, \Delta^+)$, en DeLP se asume que el conjunto Π es un conjunto no contradictorio, y

por lo tanto no se pueden derivar estrictamente (*esto es,* usando solamente Π) literales complementarios.

En vista de responder consultas al sistema DeLP utiliza como procedimiento de prueba un mecanismo de análisis global para decidir qué literales fijos serán aceptados a partir de un programa. Como fue brevemente explicado con anterioridad, tal procedimiento está basado en el uso de argumentos para un literal fijo. En base a los argumentos construidos se efectúa un análisis dialéctico que permite decidir cuando un literal fijo está aceptado considerando los posibles argumentos de manera recurrente.

Intuitivamente, un argumento funciona como una razón a favor de creer en la validez del literal que éste soporta, de forma que $\langle \mathcal{A}, L \rangle$ es un argumento para el literal L a partir de $\mathcal{P}$, si $\mathcal{A}$ es un conjunto de reglas rebatibles y presuposiciones de Δ^+, tal que:

1. $\Pi \cup \mathcal{A} \vdash\!\sim L$,
2. $\Pi \cup \mathcal{A}$ es no contradictorio, y
3. $\mathcal{A}$ es minimal: no existe $\mathcal{A}' \subset \mathcal{A}$ tal que satisface las condiciones (1) y (2).

Por conveniencia a veces llamaremos a un argumento $\langle \mathcal{A}, L \rangle$, simplemente un argumento $\mathcal{A}$ para L. Además, un argumento $\langle \mathcal{B}, Q \rangle$ es un subargumento de un argumento $\langle \mathcal{A}, L \rangle$ si $\mathcal{B} \subseteq \mathcal{A}$.

Ejemplo 2.4 *A partir del programa $\mathcal{P}$ es posible construir los argumentos $\langle \mathcal{B}_1, \sim comprar\_acciones(acme) \rangle$ y $\langle \mathcal{B}_2, comprar\_acciones(acme) \rangle$, donde*

$$\mathcal{B}_1 = \left\{ \begin{array}{l} \sim comprar\_acciones(acme) \prec buen\_precio(acme), empresa\_riesgosa(acme) \\ empresa\_riesgosa(acme) \prec huelga\_empleados(acme) \\ huelga\_empleados(acme) \prec \end{array} \right\}$$

$$\mathcal{B}_2 = \left\{ \; comprar\_acciones(acme) \prec buen\_precio(acme) \; \right\}$$

Dos literales L y L_1 están en desacuerdo con respecto a un programa (Π, Δ^+), si $\Pi \cup \{L, L_1\}$ es contradictorio. De esta manera, diremos que dos argumentos estarán en conflicto cuando sustenten conclusiones en desacuerdo. Esto es, el argumento $\langle \mathcal{A}_1, L_1 \rangle$ contra-argumenta o ataca al argumento $\langle \mathcal{A}_2, L_2 \rangle$ en el literal L, si y solo si existe un subargumento $\langle \mathcal{A}, L \rangle$ de $\langle \mathcal{A}_2, L_2 \rangle$ tal que L y L_1 están en desacuerdo. El argumento $\langle \mathcal{A}, L \rangle$ se llama subargumento de desacuerdo, y el literal L será el punto de contra-argumentación.

La noción de contra-argumento determina cuando un argumento ataca a otro, capturando la noción de conflicto entre argumentos a través de la negación fuerte. Observe, por ejemplo, que considerando los argumentos $\langle \mathcal{B}_1, \sim comprar\_acciones(acme) \rangle$ y $\langle \mathcal{B}_2, comprar\_acciones(acme) \rangle$ del Ejemplo 2.4, $\langle \mathcal{B}_1, \sim comprar\_acciones(acme) \rangle$ es un contra-argumento de $\langle \mathcal{B}_2, comprar\_acciones(acme) \rangle$. Es posible identificar dos situaciones de ataque entre argumentos: el *ataque directo* cuando un argumento ataca a otro directamente a su conclusión, y el *ataque interno* que ocurre cuando un argumento ataca a un subargumento propio del argumento atacado. Para decidir si un ataque realmente tiene éxito y constituye una derrota se necesita de un *criterio para comparar argumentos* que establezca cuando un argumento es preferido a otro. Por lo general, en la literatura de programación en lógica rebatible para modelar un criterio de comparación de argumentos asumiremos una relación de preferencia, denotada $\succeq$, la cual será utilizada para definir un orden de preferencia entre dos argumentos.

El orden definido por una relación de preferencia dependerá de las propiedades que se asume para dicha relación, por lo tanto como la comparación entre argumentos podría definirse de varias maneras, en lo que sigue asumiremos que existe un criterio que define un orden parcial estricto "$\succ$" que permite distinguir cuando un argumento es preferido a otro. Por ejemplo, si $\langle \mathcal{A}_1, L_1 \rangle$ es preferido a $\langle \mathcal{A}_2, L_2 \rangle$, se lo denotará como $\langle \mathcal{A}_1, L_1 \rangle \succ \langle \mathcal{A}_2, L_2 \rangle$. En general, en la comparación de dos argumentos en conflicto, puede darse uno de los siguiente cuatro casos:

- $\langle \mathcal{A}_1, L_1 \rangle$ es preferido a $\langle \mathcal{A}_2, L_2 \rangle$, denotado $\langle \mathcal{A}_1, L_1 \rangle \succ \langle \mathcal{A}_2, L_2 \rangle$.
- $\langle \mathcal{A}_2, L_2 \rangle$ es preferido a $\langle \mathcal{A}_1, L_1 \rangle$, denotado $\langle \mathcal{A}_2, L_2 \rangle \succ \langle \mathcal{A}_1, L_1 \rangle$.
- $\langle \mathcal{A}_1, L_1 \rangle$ y $\langle \mathcal{A}_2, L_2 \rangle$ son equivalentes, denotado $\langle \mathcal{A}_1, L_1 \rangle \approx \langle \mathcal{A}_2, L_2 \rangle$.
- $\langle \mathcal{A}_1, L_1 \rangle$ y $\langle \mathcal{A}_2, L_2 \rangle$ son incomparables, denotado $\langle \mathcal{A}_1, L_1 \rangle \bowtie \langle \mathcal{A}_2, L_2 \rangle$.

En DeLP el criterio puede ser reemplazado de forma modular, por lo que se puede utilizar el criterio más adecuado al problema que se intenta modelar. En base a los posibles resultados de la comparación de dos argumentos podemos definir a la derrota entre argumentos, donde el argumento $\langle \mathcal{A}_1, L_1 \rangle$ es un derrotador para el argumento $\langle \mathcal{A}_2, L_2 \rangle$ en el literal L, si y solo si existe un subargumento $\langle \mathcal{A}, L \rangle$ de $\langle \mathcal{A}_2, L_2 \rangle$ tal que $\langle \mathcal{A}_1, L_1 \rangle$ contra-argumenta a $\langle \mathcal{A}_2, L_2 \rangle$ en el literal L, y vale que:

- $\langle \mathcal{A}_1, L_1 \rangle$ es preferido a $\langle \mathcal{A}, L \rangle$ de acuerdo al criterio de comparación utilizado ($\langle \mathcal{A}_1, L_1 \rangle \succ \langle \mathcal{A}, L \rangle$), denotando que el argumento $\langle \mathcal{A}_1, L_1 \rangle$ es un derrotador propio para el argumento $\langle \mathcal{A}_2, L_2 \rangle$; o

- $\langle \mathcal{A}_1, L_1 \rangle$ no es mejor que $\langle \mathcal{A}, L \rangle$, ni $\langle \mathcal{A}, L \rangle$ es mejor que $\langle \mathcal{A}_1, L_1 \rangle$ de acuerdo al criterio de comparación utilizado, lo que significa que el argumento $\langle \mathcal{A}_1, L_1 \rangle$ es un derrotador por bloqueo para el argumento $\langle \mathcal{A}_2, L_2 \rangle$.

Podemos utilizar la noción de derrota para saber si un argumento es aceptado, y por lo tanto si el literal soportado por este es creído. Por ejemplo, si a partir de un programa $\mathcal{P}$ se obtiene un argumento $\langle \mathcal{A}_0, L_0 \rangle$ que no tiene derrotadores, entonces un agente que disponga de $\mathcal{P}$ para razonar, podrá "creer" en el literal L_0. Sin embargo, puede ocurrir que $\langle \mathcal{A}_0, L_0 \rangle$ tenga un derrotador $\langle \mathcal{A}_1, L_1 \rangle$, con lo cual existirán razones para invalidar la creencia en L_0. Pero también puede ocurrir que la estructura $\langle \mathcal{A}_1, L_1 \rangle$ tenga a su vez un derrotador $\langle \mathcal{A}_2, L_2 \rangle$, reinstaurando la creencia en L_0. La secuencia de interacciones puede ir más lejos, y a su vez puede existir un derrotador $\langle \mathcal{A}_3, L_3 \rangle$ para $\langle \mathcal{A}_2, L_2 \rangle$, que vuelve a invalidar la creencia en L_0, y así siguiendo. Lo cual da origen una secuencia de argumentos [$\langle \mathcal{A}_0, L_0 \rangle$, $\langle \mathcal{A}_1, L_1 \rangle$, $\langle \mathcal{A}_2, L_2 \rangle$, $\langle \mathcal{A}_3, L_3 \rangle$,...] que se llamará línea de argumentación, donde cada elemento es un derrotador de su predecesor.

En una línea argumentativa para un argumento $\langle \mathcal{A}_1, L_1 \rangle$ los argumentos en posiciones impares juegan el rol de argumentos de soporte (*esto es,* están a favor de $\langle \mathcal{A}_1, L_1 \rangle$), y los argumentos en posiciones pares actúan como argumentos de interferencia (*esto es,* actúan en contra de $\langle \mathcal{A}_1, L_1 \rangle$). Para evitar situaciones falaces [GS04], en DeLP es necesario contar con *líneas de argumentación aceptables*. Una línea de argumentación Λ es aceptable si cumple las siguientes condiciones: (1) Λ es finita, (2) el conjunto de argumentos de soporte en Λ es no contradictorio y el conjunto de argumentos de interferencia en Λ también es no contradictorio, (3) ningún argumento $\mathcal{A}_j$ en Λ es un subargumento de un argumento $\mathcal{A}_i$ en Λ, $i < j$, y (4) todo derrotador por bloqueo $\mathcal{A}_i$ en Λ es derrotado por un derrotador propio $\mathcal{A}_{i+1}$ en Λ.

Claramente un argumento podría tener varios derrotadores a partir de un programa $\mathcal{P}$. La presencia de múltiples derrotadores para un mismo argumento produce una ramificación de líneas de argumentación, dando origen a un estructura de árbol de derrotadores. En DeLP esta estructura se denomina *árbol de dialéctica*. Un árbol de dialéctica para $\langle \mathcal{A}_0, L_0 \rangle$, a partir de $\mathcal{P}$, se denota $\mathcal{T}_{\langle \mathcal{A}_0, L_0 \rangle}$, y se construye de forma tal que cada camino

desde la raíz a una hoja corresponde a una línea de argumentación aceptable. De esta forma todo nodo (excepto el raíz) es un derrotador de su padre, y las hojas son argumentos no derrotados.

Una vez que el árbol está construido debemos establecer una forma de saber si el literal en la raíz es creído o no en base al conocimiento expresado en el programa DeLP. Esto es logrado a través del marcado de un árbol de dialéctica, el cual es un proceso que marca cada nodo como derrotado "D" o no derrotado "U", de la siguiente forma: los nodos hojas son marcados como "U"; y, un nodo interno es marcado como "D" si tiene al menos un hijo marcado como "U", o como "U" si todos sus hijos están marcados como "D". Un árbol de dialéctica marcado $\mathcal{T}^*_{\langle \mathcal{A}, L\rangle}$ representa el análisis dialéctico en el cual todos los argumentos construibles a partir de un programa $\mathcal{P}$ que se relacionan con el análisis son considerados a fin de decidir el estado de un argumento $\langle \mathcal{A}, L\rangle$. Por lo tanto, el estado de un argumento $\langle \mathcal{A}, L\rangle$ será "garantizado" si la raíz tiene como marca "U". A su vez, un literal L está garantizado en $\mathcal{P}$ (y es por lo tanto creído en base al conocimiento del programa) si y solo si existe un argumento $\langle \mathcal{A}, L\rangle$ para L y $\langle \mathcal{A}, L\rangle$ está garantizado en $\mathcal{P}$. Esto es, sea L un literal que representa una consulta para un programa $\mathcal{P}$. La respuesta a L será:

- SI: si L está garantizado en $\mathcal{P}$.
- NO: si $\overline{L}$ está garantizado en $\mathcal{P}$.
- INDECISO: si L no está garantizado en $\mathcal{P}$y $\overline{L}$ no está garantizado en $\mathcal{P}$.
- DESCONOCIDO: si L no pertenece al lenguaje del programa $\mathcal{P}$.

2.2.4. Datalog$^{\pm}$

Finalmente, otro lenguaje que ha ganado popularidad en los últimos tiempos es Datalog$^{\pm}$ [CGL12, CGK08, CGK13]. Esto viene aparejado con el hecho de que las ontologías y los sistemas basados en reglas juegan un papel preponderante en el desarrollo de entornos colaborativos en los cuales se busca que la semántica detrás de los contenidos sea entendible por las computadoras, tratando de esta forma que los mismos sean procesables por éstas últimas minimizando lo más posible la necesidad de influencia humana. Esto es especialmente evidente en tecnologías como la Web Semántica [BLHL01]. En tal contexto,

en los últimos tiempos la investigación en tales campos se ha centrado en la definición y el desarrollo de formalismos con alta capacidad de escalabilidad para ser utilizados en la Web de Datos (Web of Data), principalmente mediante la explotación de tecnologías de Bases de Datos. Para conseguir esto último es particularmente importante definir formas de generalizar reglas de bases de datos y dependencias funcionales de forma tal que puedan expresar axiomas ontológicos.

Para paliar tal problemática se ha desarrollado la familia Datalog$^{\pm}$ de variantes de Datalog, los cuales extienden el Datalog básico con la posibilidad de utilizar cuantificaciones existenciales en las cabezas de las reglas, entre otras características importantes, manteniendo a su vez una tratabilidad adecuada para entornos de alto volumen de información mediante restricciones a la sintaxis de reglas. Datalog$^{\pm}$ permite un estilo modular de representación de conocimiento basado en reglas, lo que lo hace útil en diversos entornos prácticos, como ser resolución de consultas en ontologías (ontology querying), extracción de datos de la Web (Web data extraction) o intercambio de datos (data exchange) [LMS12]. Esta característica de la familia de lenguajes Datalog$^{\pm}$ de lograr un buen balance entre tratabilidad y expresividad hace que la misma sea muy atrayente a la hora de definir una infraestructura que permita la realización de procesos de argumentación masiva sobre grandes volúmenes de datos almacenados en repositorios federados, principalmente bases de datos (el cual es un objetivo a largo plazo apoyado por la presente investigación). El aporte de Datalog$^{\pm}$ a tal objetivo es doble: por un lado, su expresividad nos permitirá expresar los esquemas de los repositorios que serán parte de la federación; mientras que por el otro su tratabilidad en el manejo de grandes volúmenes de datos será de gran utilidad en caso de necesitarse representar los datos almacenados en las bases de datos a través de las ontologías.

Por todo lo mencionado es que Datalog$^{\pm}$ es central a la presente disertación. Como tal, el mismo será presentado en un capítulo separado, donde nos podamos enfocar tanto en los aspectos de representación de conocimiento del mismo como así también en los procesos utilizados en la resolución de consultas en el lenguaje, y otros conceptos de gran importancia en el entorno de consolidación de ontologías como ser inconsistencia e incoherencia.

2.3. Resumen

En este capítulo hemos hecho una breve introducción a la representación de conocimiento y el razonamiento, presentando tanto los lenguajes más tradicionales del área como así también lenguajes más recientes que han sido desarrollados pensando en su utilización en entornos colaborativos. La representación del conocimiento y el razonamiento es un área de la Inteligencia Artificial cuyo objetivo fundamental es expresar conocimiento acerca de un dominio de forma tal que facilite la inferencia (sacar conclusiones, hacer explícita información implícita) a partir de dicho conocimiento, de forma tal que problemas complejos puedan ser resueltos.

Dentro de los lenguajes más difundidos en KR&R se encuentra la Lógica Proposicional. La misma es un sistema formal cuyos elementos más simples representan proposiciones, y cuyas constantes lógicas, representan operaciones sobre proposiciones, capaces de formar otras proposiciones de mayor complejidad. El razonamiento en la lógica proposicional es logrado a través de un mecanismo que primero evalúa sentencias simples y luego sentencias complejas. Este mecanismo determina la veracidad de una sentencia compleja, analizando los valores de veracidad asignados a las sentencias simples que la conforman. De esta forma, intuitivamente podemos afirmar que la semántica nos informa el significado de las proposiciones en el mundo real. En este sentido, las conectivas generan un significado dentro de las proposiciones compuestas, a partir de las proposiciones componentes que conectan.

Otro lenguaje ampliamente difundido es la Lógica de Primer Orden o Lógica de Predicados, el cual surge para agregar mayor expresividad a la Lógica Proposicional, permitiendo así representar enunciados coloquiales más complejos. La estrategia básica para lograr esto es la utilización de cuantificadores y de variables interpretables como entidades, algo que no es posible en la Lógica Proposicional. La semántica detrás de la Lógica de Primer Orden es similar a aquella de la Lógica Proposicional, de forma tal que nos permite saber el valor de verdad de una fórmula, donde la misma se determina en dos etapas: atribuyendo una interpretación a cada término utilizado precisando la entidad del universo modelado; y asignando un valor de verdad a los predicados según si la relación denotada se verifica o no en el universo. De esta manera, conociendo los valores de verdad de las fórmulas atómicas, se puede determinar el valor de verdad de una fórmula compuesta aplicando la semántica convencional de los símbolos lógicos.

Sin embargo, si bien la Lógica Proposicional y la Lógica de Predicados han sido ampliamente difundidos y utilizados, los mismos no suelen ser adecuados para su utilización en entornos colaborativos masivos. El principal problema al respecto es el trade-off que suele ser reconocido en KR&R: generalmente, mayor expresividad implica mayor complejidad en los procesos de inferencia y determinación de verdad. En general, en entornos colaborativos masivos un aspecto crucial es el manejo de grandes cantidades de datos, junto con la tolerancia a las inconsistencias que puedan surgir a partir del mantenimiento colaborativo de las bases de conocimiento. Es por esta razón que en los últimos años han surgido con cada vez mayor fuerza nuevos formalismos enfocados en su utilización en tales entornos, donde en general se limita hasta cierto punto la expresividad de los lenguajes en pos de obtener mayor eficiencia. Dentro de tales lenguajes podemos encontrar a las Lógicas Descriptivas, la Programación Lógica y la Programación Lógica Rebatible, así como también la familia de lenguajes en los que se enfocan los desarrollos de la presente tesis, Datalog$^{\pm}$, la cual se encuentra influenciada en cierto modo por desarrollos previos en DL.

Las lógicas descriptivas (Description Logics - DL) son un conjunto de lenguajes de representación, los cuales pueden utilizados tanto para representar conocimiento acerca de individuos particular como así también de conceptos y relaciones entre éstos; de una forma estructurada y formalmente bien comprendida. Como es característico en los lenguajes ontológicos, una base de conocimiento en DL está formada por dos componentes, donde uno de ellos se enfoca en la descripción de elementos terminológicos (TBox) mientras que el otro se encarga de expresar conocimiento particular acerca de individuos (ABox) por lo que es a veces denominado *descripción del mundo (world description)*. Luego, la semántica de una lógica descriptiva es definida mediante la interpretación de conceptos como conjuntos de individuos (aquellos individuos que corresponden al concepto) y la interpretación de roles como pares ordenados de individuos (aquellos relacionados a través del rol). Tales individuos son asumidos para un dominio dado. A su vez, la semántica de conceptos y roles no atómicos es definida en términos de aquellos que sí son atómicos, mediante recursiones.

Dos conceptos cruciales para la presente tesis son aquellos de inconsistencia e incoherencia. Primeramente, como es esperable definimos a una base de conocimiento en DL KB como consistente si la misma es tal que tiene una interpretación posible, e inconsistente sino. Es decir, KB es consistente si y sólo si existe $\mathcal{I}$ tal que $\mathcal{I} \models KB$. Respecto, de incoherencia, tal concepto es definido en términos de satisfacibilidad de conceptos:

intuitivamente, un concepto es satisfacible si el mismo *tiene sentido*, no es contradictorio; es decir, existe una interpretación donde los axiomas de la TBox son satisfechos y el concepto en cuestión no implica un conjunto vacío (*esto es,* hay al menos un individuo que corresponde al concepto). Si esta situación no es posible decimos que el concepto es insatisfacible. En base a la distinción entre conceptos satisfacibles e insatisfacibles podemos definir incoherencia para una ontología DL: decimos que KB es incoherente si tiene al menos un concepto insatisfacible en su TBox [FHP+06]. Como veremos más adelante en la tesis, los problemas de inconsistencia e incoherencia son muy importantes a la hora de utilizar el conocimiento, especialmente en entornos colaborativos, por lo que deben ser atacados o manejados de alguna forma, lo cual es una de los objetivos principales del desarrollo que aquí presentaremos.

A continuación, en el capítulo hemos presentado otros dos formalismos de KR&R, los cuales utilizan fragmentos de la Lógica de Primer Orden en la representación de conocimiento. A saber, estos son la Programación Lógica, y la Programación Lógica Rebatible. En estos formalismos el conocimiento se expresa mediante un conjunto finito de axiomas y las hipótesis del problema a resolver, mientras que una sentencia meta que representa el objetivo que queremos alcanzar. La gran diferencia entre ambos enfoques es que la Programación Lógica Rebatible cambia la forma en que las respuestas son obtenidas a partir de un programa, para de esta forma hacer que el formalismo sea tolerante a inconsistencias, algo que no sucede con la Programación Lógica. La estrategia básica es utilizar para cada consulta efectuada al sistema un análisis dialéctico basado en el uso de argumentos, que son razones a favor o en contra de aquello que queremos probar. Intuitivamente, tal proceso dialéctico involucra el hecho de que un argumento que sustenta una conclusión L puede ser atacado por otros argumentos *derrotadores* que lo contradigan y que pueden construirse a partir del programa. Dichos derrotadores podrán a su vez ser atacados, y así sucesivamente, generando una secuencia de argumentos llamada *línea de argumentación*. Cada línea de argumentación satisface ciertas propiedades, a fin de evitar que se generen argumentaciones falaces. El proceso completo considera para cada argumento todos sus posibles argumentos derrotadores, que en lugar de una única línea de argumentación, genera un conjunto de líneas que son representadas de una manera compacta por medio de un *árbol de dialéctica*. Este análisis dialéctico determinará si la conclusión en cuestión está *garantizada* o no, a partir del programa, y tal análisis es llevado a cabo de forma tal que en el caso de inconsistencias a lo sumo una de las piezas en conflicto es garantizada.

Finalmente, otro formalismo de KR&R que ha ganado importancia en los últimos tiempos es la familia Datalog$^{\pm}$ de variantes de Datalog, los cuales extienden el Datalog básico con la posibilidad de utilizar cuantificaciones existenciales en las cabezas de las reglas, entre otras características importantes, manteniendo a su vez una tratabilidad adecuada para entornos de alto volumen de información mediante restricciones a la sintaxis de reglas. Datalog$^{\pm}$ permite un estilo modular de representación de conocimiento basado en reglas, lo que lo hace útil en diversos entornos prácticos. En el siguiente capítulo de la disertación procedemos a introducir apropiadamente Datalog$^{\pm}$, desde cómo el conocimiento es expresado y las inferencias realizadas hasta cómo los conceptos de inconsistencia e incoherencia surgen y afectan al mismo.

Capítulo 3

El lenguaje ontológico Datalog$^{\pm}$

Las ontologías y los sistemas basados en reglas juegan un papel preponderante en el desarrollo de entornos colaborativos donde el significado de sus contenidos es procesable por computadoras, como la Web Semántica [BLHL01]. En tal contexto, en los últimos tiempos la investigación en tales campos se ha centrado en la definición y el desarrollo de formalismos con alta capacidad de escalabilidad para ser utilizados en la Web de Datos (Web of Data), principalmente mediante la explotación de tecnologías de Bases de Datos. Para conseguir esto último es particularmente importante definir formas de generalizar reglas de bases de datos y dependencias funcionales de forma tal que puedan expresar axiomas ontológicos.

Para paliar tal problemática se ha desarrollado la familia Datalog$^{\pm}$ de variantes de Datalog, los cuales extienden el Datalog básico con la posibilidad de utilizar cuantificaciones existenciales en las cabezas de las reglas, entre otras características importantes, manteniendo a su vez una tratabilidad adecuada para entornos de alto volumen de información mediante restricciones a la sintaxis de reglas. Datalog$^{\pm}$ permite un estilo modular de representación de conocimiento basado en reglas, lo que lo hace útil en diversos entornos prácticos, como ser resolución de consultas en ontologías (ontology querying), extracción de datos de la Web (Web data extraction) o intercambio de datos (data exchange) [LMS12]. En particular, uno de los objetivos a largo plazo de la línea de investigación que anidó el desarrollo de la presente tesis es la definición de una infraestructura que permita la realización de procesos de argumentación masiva sobre grandes volúmenes de datos almacenados en repositorios federados, un objetivo en el cual un lenguaje con la escalabilidad de Datalog$^{\pm}$ puede resultar de mucha ayuda, tanto proveyendo las bases de

un mecanismo de recuperación de información sobre el cual se apoye el razonamiento argumentativo como dando herramientas que asistan en la creación de repositorios federados apoyados en otras tecnologías como las bases de datos relacionales mediante la utilización del enfoque de global-como-vista (global-as-view [CDL01, CDL01, GCS08]) donde una ontología global pueda ser utilizada como una vista de los bases de datos locales que almacenen propiamente los datos.

En el presente capítulo presentaremos las características más relevantes de Datalog$^{\pm}$, introduciendo a su vez los elementos más importantes del mismo, lo que nos dará el marco de trabajo para los operadores de consolidación de ontologías Datalog$^{\pm}$ que se presentarán en capítulos posteriores de la tesis. En la primer parte del capítulo trataremos diversos conceptos tradicionales de la literatura en Datalog$^{\pm}$ que hacen a la representación de conocimiento en el lenguaje, para así contribuir a hacer la tesis autocontenida. Como tal, esos primeros conceptos no constituyen un aporte original de la presente tesis, sino que son recapitulaciones de trabajos previos; en particular, la introducción de los componentes de Datalog$^{\pm}$ y la resolución de consultas en el formalismo es tomada de los trabajos de Calí *et al.* [CGL12, CGK08, CGK13]. Luego, en la segunda parte del capítulo se procede a analizar los distintos problemas que pueden surgir en las ontologías Datalog$^{\pm}$. La naturaleza del contenido en esta segunda parte es doble: por un lado, la definición del concepto de inconsistencia de ontologías Datalog$^{\pm}$ es tomado de la literatura, en particular de [LMS12], solamente proveyendo la tesis de ejemplos originales; por el otro, el segundo problema mencionado, incoherencia, es uno que no ha sido tratado en la literatura, y por lo tanto corresponde a un aporte originmal de la presente tesis, tanto a nivel de definición del problema como en los ejemplos presentados en su tratado.

3.1. Conceptos básicos de Datalog$^{\pm}$

A continuación introduciremos los elementos básicos de la familia Datalog$^{\pm}$ de extensiones a Datalog. Comenzaremos por introducir algunas nociones básicas que serán utilizadas por el resto de la tesis y que nos permitirán definir los elementos que componen a una ontología Datalog$^{\pm}$, y que por lo tanto serán sobre los cuales operan los operadores de consolidación que se introducirán en capítulos posteriores.

3.1.1. Nociones básicas

Comenzaremos por presentar las nociones básicas sobre las que se apoyan las definiciones de los elementos en las ontologías Datalog$^{\pm}$. Por razones de simplicidad se dará una mera introducción a los mismos, un tratado más amplio puede encontrarse en [CGL12].

En la presente tesis asumiremos que el dominio del discurso de una ontología Datalog$^{\pm}$ consiste en un conjunto contable de *constantes de datos* Δ, un conjunto contable de elementos nulos Δ_N (los cuales se utilizarán para representar valores desconocidos), y un conjunto contable de variables $\mathcal{V}$. Además, asumiremos que el dominio del discurso respeta la *presunción de nombre único*: esto es, que diferentes constantes en Δ representan distintos valores. Respecto de la notación, para distinguir las constantes de las variables adoptamos la notación estándar en Programación Lógica, donde los nombres de variables comienzan con letras mayúsculas, y donde constantes y símbolos de predicados comienzan con letras minúsculas.

También asumimos la existencia de un *esquema relacional* $\mathcal{R}$, el cual es un conjunto finito de símbolos de predicado (o simplemente predicados). Un *término* t es una constante, un nulo o una variable. Un *átomo* a tiene la forma $\mathrm{p}(t_1, \ldots, t_n)$, donde p es un predicado n-ario y $t_1, \ldots, t_n$ son términos; un átomo es completamente instanciado o fijo (ground) si y sólo si todos sus términos son constantes. Sea $\mathcal{L}$ un lenguaje de primer orden, y $\mathcal{R} \subset \mathcal{L}$; denotaremos con $\mathcal{L}_{\mathcal{R}}$ el sublenguaje generado por $\mathcal{R}$. Un *homomorfismo* sobre constantes, nulos y variables es un mapeo $h : \Delta \cup \Delta_N \cup \mathcal{V} \longrightarrow \Delta \cup \Delta_N \cup \mathcal{V}$ tal que (i) $c \in \Delta$ implica que $h(c) = c$, (ii) $c \in \Delta_N$ implica que $h(c) \in \Delta \cup \Delta_N$, y (iii) h es extendido de manera natural a átomos, conjuntos de átomos, y conjunciones de átomos.

3.1.2. Representación de información en Datalog$^{\pm}$

Ahora que hemos introducido las nociones básicas procederemos a introducir los elementos que permiten representar conocimiento en Datalog$^{\pm}$. Para tal fin, en Datalog$^{\pm}$ se utiliza una combinación de cuatro componentes: un conjunto de átomos conocido como *base de datos*, un conjunto de reglas que permiten obtener nueva información a partir de las mismas conocidas como *tuple-generating dependencies*, restricciones sobre los átomos que pueden existir en el universo representadas a través de *negative constraints* y reglas que imponen la igualdad entre ciertos átomos conocidas como *equality-generating dependencies*.

Base de Datos - Componente D

El componente más sencillo a través del cual se puede representar conocimiento en Datalog$^{\pm}$ son las bases de datos. Intuitivamente, una base de datos D es un conjunto de átomos que representan hechos conocidos acerca del mundo.

De manera más formal, dado un esquema relacional $\mathcal{R}$ y un conjunto de constantes Δ, decimos que D es una base de datos (instancia) de $\mathcal{R}$ si D es un conjunto finito de átomos con predicados pertenecientes a $\mathcal{R}$ y términos pertenecientes a $\Delta \cup \Delta_N$.

Ejemplo 3.1 (Base de datos D) *Supongamos que estamos queriendo representar información acerca de los gustos y habilidades musicales de tres amigos: Ronnie, Axl y Simone. Entonces, una posible base de datos es la siguiente:*

$$D = \{cantante\_rock(axl), buen\_cantante(simone), miembro\_banda\_pop(ronnie),$$
$$fanatico(axl, banda_1), estilo(banda_1, metal), toca(ronnie, guitarra), mal\_cantante(ronnie)\}$$

En esta base de datos podemos ver como se representa a través de átomos distintos hechos que conocemos acerca del mundo, como por ejemplo que Simone canta bien y que Ronnie sabe tocar la guitarra.

Tuple-generating dependencies - TGDs

Las dependencias de generación de tuplas (TGDs, por su nombre en inglés tuple-generating dependencies) son reglas que pueden ser utilizadas para obtener nuevo conocimiento (nuevos átomos) a partir de un conjunto de átomos. Por lo tanto, las TGDs son un componente esencial para la representación de conocimiento en el formalismo. A continuación presentamos las características mas importantes de estas dependencias.

Dado un esquema relacional $\mathcal{R}$, una TGD σ es una fórmula de primer orden de la forma $\forall \mathbf{X} \forall \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z})$ donde $\Phi(\mathbf{X}, \mathbf{Y})$ y $\Psi(\mathbf{X}, \mathbf{Z})$ son conjunciones de átomos sobre $\mathcal{R}$ llamados el *cuerpo* de la TGD (denotado por $cuerpo(\sigma)$) y la *cabeza* (denotada $cabeza(\sigma)$), respectivamente. Decimos que una TGD σ es *aplicable* para una base de datos D si y solo si existe un homomorfismo h que mapea los átomos en $\Phi(\mathbf{X}, \mathbf{Y})$ a átomos en D. Adicionalmente, una TGD aplicable Fσ es *satisfecha* por una base de datos D para $\mathcal{R}$ si y sólo si existe una extensión h' de h que mapea los átomos de $\Psi(\mathbf{X}, \mathbf{Z})$ a átomos de D.

Cabe destacar que evaluar consultas conjuntivas booleanas (boolean conjunctive queries - BCQ-eval) para un conjunto Σ_T de TGDs es equivalente a evaluar consultas conjuntivas booleanas para un conjunto Σ'_T, construido a partir de Σ_T, de TGDs con un sólo átomo en sus cabezas (para ver esto en más profundidad referimos al lector al Lema 10 en [CGK08]); por lo tanto, y sin pérdida de generalidad, a partir de este punto asumiremos que las TGDs tienen un sólo átomo en su cabeza [CGK13]. A su vez, es importante aclarar que por simplicidad notacional por lo general omitiremos los cuantificadores en las TGDs.

Como dijimos anteriormente, una de las características más importantes y atractivas de la familia de lenguajes Datalog$^{\pm}$ es su decidibilidad. La principal condición (sintáctica) de las TGDs que garantiza la decidibilidad de la evaluación de consultas conjuntivas son resguardo (guardedness) [CGK13], adherencia (stickiness) [CGP12] y aciclicidad. Una TGDs σ es llamada resguardada si existe un átomo a en $cuerpo(\sigma)$ que contiene (o "resguarda") todos las variables del cuerpo de σ. Una subclase de TGDs resguardadas son las llamadas TGDs lineales que tienen un sólo átomo en el cuerpo (el cual es automáticamente el resguardo). Adherencia es inherentemente diferente de resguardo, y su propiedad fundamental puede ser descripta informalmente como sigue: variables que aparecen más de una vez en el cuerpo, las cuales son variables de concatenación (join), son siempre propagadas a los átomos inferidos a través de las TGDs. Finalmente, un conjunto de TGDs Σ es llamado acíclico si su grafo de predicados es acíclico; un conjunto acíclico de TGDs puede ser visto como un conjunto no recursivo de TGDs. Cada una de estas condiciones tiene una versión "débil": resguardo debilitado [CGK13], adherencia debilitada [CGP12] y aciclicidad debilitada [FKMP03, FKMP05], respectivamente. Es importante aclarar que los formalismos introducidos en la presente disertación pueden ser aplicados a conjuntos no restringidos de TGDs; sin embargo, durante la disertación asumiremos ontologías que pertenezcan a fragmentos tratables de Datalog$^{\pm}$, tales como los presentados recientemente.

Ejemplo 3.2 (TGDs) *A continuación mostramos un ejemplo del tipo de conocimiento que puede ser expresado a través de tuple-generating dependencies. Consideremos nuevamente el Ejemplo 3.1 acerca de las habilidades y gustos musicales de Ronnie, Axl y Simone.*

En este entorno podríamos querer expresar, por ejemplo, que a todo aquel fanático de cierta banda le gusta el estilo musical al cual la banda subscribe. Esto sería representado a través de la TGD

$$\sigma = fanatico(Persona, Banda) \wedge estilo(Banda, Estilo) \rightarrow le\_gusta(Persona, Estilo)$$

.

Negative Constraints - NCs

Otro componente utilizado en Datalog$^{\pm}$ para la representación de conocimiento son las restricciones negativas (NCs, por su nombre en inglés negative constraints), las cuales intuitivamente son reglas que se utilizan para expresar relaciones que no pueden darse entre átomos.

Formalmente, las NCs son fórmulas de primer orden de la forma $\forall \mathbf{X}\Phi(\mathbf{X}) \rightarrow \perp$,las cuales tienen por cuerpo $\mathbf{X}$ una conjunción de átomos y por cabeza tienen la constante de verdad lógica *falso*, denotada $\perp$. Por lo tanto, para no ser violadas estas restricciones deben evaluar como falsas para una base de datos D bajo un conjunto de TGDs $\Sigma_{_T}$. Esto es, una NC τ está satisfecha para una base de datos D bajo $\Sigma_{_T}$ si y sólo si no existe un homomorfismo h que mapea los átomos de $\Phi(\mathbf{X})$ a D, donde D es tal que toda TGD en $\Sigma_{_T}$ esta satisfecha. Como veremos a través de la presente disertación, las NCs son importantes a la hora de identificar inconsistencias en una ontología Datalog$^{\pm}$, ya que la violación de las mismas es una de las principales fuentes de conflictos de consistencia. Por lo tanto, las NCs son un ingrediente importante a ser considerado en cualquier proceso de consolidación como los introducidos en este trabajo.

Nuevamente, en general omitiremos los cuantificadores universales en las NCs.

Ejemplo 3.3 (NCs) *Continuando con el entorno presentado en el Ejemplo 3.1, podemos representar el hecho de que una persona no puede ser al mismo tiempo un buen y un mal cantante a través de la NC*

$$\tau = buen\_cantante(Persona) \wedge mal\_cantante(Persona) \rightarrow \perp$$

Equality-generating dependencies - EGDs

Finalmente, el último componente utilizado para representar conocimiento en Datalog$^{\pm}$ son las dependencias de generación de igualdad (EGD, por su nombre en inglés equality-generating dependencies). Estas tienen una doble función: por un lado, las mismas sirven para chequear que no existen dos átomos en el componente D de una ontología Datalog$^{\pm}$ que unifiquen con distintos valores a las variables restringidas por una EGD; por el otro, las EGD se pueden utilizar para unificar valores nulos generados a través de una TGD con valores constantes presentes en átomos en D.

Desde un punto de vista formal, las EGDs son fórmulas de primer orden de la forma $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow X_i = X_j$, donde $\Phi(\mathbf{X})$ es una conjunción de átomos, y X_i y X_j son variables de $\mathbf{X}$. Una EGD es satisfecha por una base de datos D para un esquema $\mathcal{R}$ si y sólo si cuando sea que exista un homomorfismo h tal que $h(\Phi(\mathbf{X})) \subseteq D$, entonces se verifica que $h(X_i) = h(X_j)$.

Es importante aclarar que en la presente tesis nos enfocaremos en una clase restringida de EGDs, denominadas *separables* [CGL12]; la propiedad de separabilidad de una EGD dice que, si tal EGD es violada, entonces la razón de tal violación son átomos en el componente D, y no la aplicación de alguna TGD,*esto es,* si una EGD en Σ_{E} es violada cuando aplicamos las TGDs en Σ_{T} para una base de datos D, entonces la EGD es también violada en D. Separabilidad es comúnmente asumida en ontologías Datalog$^{\pm}$, ya que una de las propiedades más importantes de esta familia es el enfocarse en fragmentos decidibles (en realidad, tratables) de Datalog$^{\pm}$.

Las EGDs también juegan un papel preponderante respecto de los conflictos que puedan aparecer en ontologías Datalog$^{\pm}$, hasta el punto de que, como veremos más adelante, los operadores introducidos en el presente trabajo deben asegurarse que, como en el caso de las NCs, ninguna EGD es violada en la ontología resultante del proceso. Nótese que la restricción de usar únicamente EGDs separables provoca que ciertos casos de conflictos no sean considerados en nuestra propuesta: el tratamiento de tales casos, si bien es interesante desde un punto de vista técnico, queda fuera del alcance del presente trabajo ya que nos enfocamos en fragmentos tratables de Datalog$^{\pm}$ que nos brinden beneficios adecuados para la definición de los procesos masivos de argumentación que son el objetivo final de la línea de trabajo que anidó la investigación aquí presentado de operadores de consolidación de ontologías Datalog$^{\pm}$. Como en los casos previos, cuando escribamos una EGD omitiremos los cuantificadores.

Ejemplo 3.4 (EGDs) *Continuando con nuestro ejemplo acerca de gustos y habilidades musicales de tres amigos, supongamos que queremos representar el conocimiento de que, en el caso particular del entorno que estamos modelando, una persona no puede tocar mas de un instrumento. Esto podemos expresarlo a través de la EGD*

$$\nu = toca(Persona, Instrum) \wedge toca(Persona, Instrum') \rightarrow Instrum = Instrum'$$

Ontologías Datalog$^{\pm}$

Hasta el momento hemos presentado aquellos componentes del lenguaje que nos permiten representar conocimiento en Datalog$^{\pm}$. Conjugando los mismos podemos obtener una ontología Datalog$^{\pm}$, que en la presente tesis nos servirá para representar el conocimiento que queremos consolidar para eliminar conflictos de consistencia o coherencia, si los hubiera. Formalmente, una ontología Datalog$^{\pm}$ es definida como sigue.

Definición 3.1 (Ontología Datalog$^{\pm}$) *Una ontología Datalog$^{\pm}$ $KB = (D, \Sigma)$, donde $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$, consiste de una base de datos D formada únicamente por átomos fijos, un conjunto de TGDs $\Sigma_{_T}$, un conjunto de EGDs separables $\Sigma_{_E}$ y un conjunto de NCs $\Sigma_{_{NC}}$.*

Como podemos ver, una ontología Datalog$^{\pm}$ tiene dos componentes separados: por un lado tenemos todos los hechos conocidos acerca del mundo en el componente D; por el otro, todo aquello que podemos utilizar para generar nuevo conocimiento a partir de D (y para restringir como el mismo es creado) se encuentra agrupado en el segundo componente, Σ. Claramente esto guarda una estrecha relación con los enfoques utilizados en otros entornos ontológicos, como ser las ABoxes y TBoxes en DLs. En la presente disertación a menos que sea aclarado explícitamente nos referiremos al conjunto Σ en una ontología Datalog$^{\pm}$ como el conjunto de restricciones de la misma, sin distinguir entre las restricciones y dependencias. Una aclaración importante que debe hacerse es que todos los conjuntos de componentes que forman una ontología Datalog$^{\pm}$ se asumen finitos.

Ejemplo 3.5 (Ontología Datalog$^{\pm}$) *Finalmente, si consideramos los ejemplos anteriores podemos construir la siguiente ontología Datalog$^{\pm}$, que muestra todo el conocimiento que queremos codificar acerca de Ronnie, Axl y Simone.*

$$KB = \left\{ \begin{array}{rl} D\text{:} & \{cantante\_rock(axl), buen\_cantante(simone), \\ & miembro\_banda\_pop(ronnie), fanatico(axl, banda_1), \\ & estilo(banda_1, metal), toca(ronnie, guitarra), \\ & mal\_cantante(ronnie)\} \\ \Sigma_{NC}: & \{\tau = buen\_cantante(P) \wedge mal\_cantante(P) \rightarrow \bot\} \\ \Sigma_{E}: & \{\nu = toca(P, I) \wedge toca(P, I') \rightarrow I = I'\} \\ \Sigma_{T}: & \{\sigma = fanatico(P, B) \wedge estilo(B, E) \rightarrow le\_gusta(P, E)\} \end{array} \right\}$$

3.1.3. Resolución de Consultas en Datalog$^{\pm}$

Una vez que tenemos una ontología, es interesante conocer cómo podemos acceder a la información en la misma (tanto explícita como implícita). Es decir, es importante como el proceso de resolución de consultas (query answering) es llevado a cabo en Datalog$^{\pm}$. A continuación daremos una introducción al respecto que nos dará el contexto necesario para entender la problemática atacada en la presente tesis; una mirada más profunda a estos conceptos puede encontrarse en [CGK13, CGL12]. Comenzaremos por brindar las bases del uso de modelos para la resolución de consultas e introduciremos además un operador de conclusión para ontologías Datalog$^{\pm}$. Finalmente, hablaremos acerca del proceso de *chase* para TGDs, el cual tiene un rol preponderante en como información implícita es obtenida a partir de aquella explícita en el componente D de una ontología.

La obtención de información a partir de una ontología Datalog$^{\pm}$ es llevada a cabo mediante la realización de consultas. Una *consulta conjuntiva* (conjunctive query) sobre $\mathcal{R}$ tiene la forma $Q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$ donde $\Phi(\mathbf{X}, \mathbf{Y})$ es una conjunción con las variables $\mathbf{X}$ e $\mathbf{Y}$, y eventualmente constantes, pero sin valores nulos. $\Phi(\mathbf{X}, \mathbf{Y})$ puede contener igualdades, pero no desigualdades. Una *consulta conjuntiva booleana* (Boolean conjunctive query) es una consulta conjuntiva de la forma $Q()$. Por lo general, una consulta conjuntiva booleana se escribe como el conjunto de todos sus átomos, teniendo como argumentos a sus constantes y variables, y omitiendo cuantificadores. La resolución de consultas se hace a través de la utilización de los homomorfismos presentados previamente. El conjunto de todas las *respuestas* para una consulta $Q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$ sobre una base de datos D,

denotado $Q(D)$, es el conjunto de todas las tuplas t sobre Δ para las cuales existe un homomorfismo $h : \mathbf{X} \cup \mathbf{Y} \mapsto \Delta \cup \Delta_N$ tal que $h(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$ y $h(\mathbf{X}) = t$. La *respuesta* a una consulta conjuntiva booleana $Q() = \exists \mathbf{Y} \Phi(\mathbf{Y})$ para una base de datos D es *Si* (Yes), denotado $D \models Q$, si y sólo si $Q(D) \neq \emptyset$, es decir, existe un homomorfismo $h : \mathbf{Y} \mapsto \Delta \cup \Delta_N$ tal que $h(\Phi(\mathbf{Y})) \subseteq D$.

Dada una base de datos D para un esquema $\mathcal{R}$ y un conjunto de restricciones $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$, el conjunto de *modelos* de D y Σ, denotado $\text{mods}(D, \Sigma)$, es el conjunto de todas las bases de datos B tal que $D \subseteq B$ y toda fórmula en Σ es satisfecha.

Ahora procederemos a describir el proceso que se utiliza para aplicar una TGD y así obtener información de una ontología mediante la aplicación sucesiva y exhaustiva de TGDs a través del *chase*, un proceso que originalmente fue utilizado para chequeo de implicación de dependencias [MMS79], y luego finalmente se aplicó a la resolución de consultas mediante el chequeo de contención de consultas [JK84].

Aplicación de TGDs. Dada una base de datos D sobre un esquema relacional $\mathcal{R}$, y una TGD σ sobre $\mathcal{R}$ de la forma $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$. Sea σ una TGD *aplicable* to D. Entonces, existe un homomorfismo h que mapea los átomos de $\Phi(\mathbf{X}, \mathbf{Y})$ a átomos en D. Sea h_1 un homomorfismo que extiende h de la siguiente forma: por cada $X_i \in \mathbf{X}$, $h_1(X_i) = h(X_i)$; por cada $Z_j \in \mathbf{Z}$, $h_1(Z_j) = z_j$, donde z_j es un valor nulo "disponible", *esto es,* $z_j \in \Delta_N$, z_j no ocurre en D, y z_j sigue en un orden lexicográfico a todos los valores nulos introducidos previamente. La *aplicación* de σ sobre D agrega a D el átomo $h_1(\Psi(\mathbf{X}, \mathbf{Z}))$ si el mismo no se encontraba ya presente en D.

En base a la aplicación de TGDs podemos definir el chase para una base de datos D y un conjunto de TGDs $\Sigma_{_T}$, denotado $\mathit{chase}(D, \Sigma_{_T})$, como la aplicación exhaustiva de las TGDs mediante primero en anchura, lo que lleva a un chase (posiblemente infinito) para D sobre $\Sigma_{_T}$, el cual puede ser utilizado para resolución de consultas en Datalog$^{\pm}$, ya que el resultado del chase de una ontología Datalog$^{\pm}$ (el cual es también referido como chase en la literatura) es un *modelo universal* de la misma [CGK08, CGK13].

3.2. Conceptos de Representación de Conocimiento y Razonamiento en Datalog$^{\pm}$

Hemos visto como los distintos componentes en una ontología Datalog$^{\pm}$ se conjugan para poder expresar el conocimiento acerca de un dominio particular. Esta capacidad expresiva sumado al manejo de grandes volúmenes de datos hacen de Datalog$^{\pm}$ un formalismo atractivo para el uso en entornos colaborativos. Sin embargo, el crecimiento de las ontologías suele traer aparejado problemas para las mismas, especialmente cuando las mismas son mantenidas cooperativamente por distintas fuentes.

A continuación presentaremos tales problemas, que son aquellos que los operadores que introduciremos más adelante se encargarán de atacar. Primeramente veremos como aparece en entornos Datalog$^{\pm}$ un concepto muy arraigado a la teoría de cambio de creencias como es el de *inconsistencia*. Luego, centraremos nuestra atención en la *incoherencia*, un fenómeno relacionado a inconsistencia que es conocido en formalismos como DLs pero que ha sido dejado de lado por la literatura Datalog$^{\pm}$. Como mostraremos en el presente capítulo, la incoherencia puede ser de gran importancia en entornos de resolución de inconsistencias, y por lo tanto no puede ser soslayada en la definición de procesos de consolidación como el que introduciremos en posteriores capítulos.

3.2.1. Inconsistencia en Datalog$^{\pm}$

La inconsistencia en entornos de explotación automática de conocimiento es un problema ampliamente reconocido como de vital importancia [GCS10, HvHH^{+}05, HvHtT05, BQL07], lo que puede verse claramente en la gran cantidad de trabajos enfocados en el manejo de la misma tanto en *Inteligencia Artificial* como en *Bases de Datos* (*v.g.,* [KP02, KM92, LM99, LS98, KP11, DJ12, BKM91, dCBB95, BS98, ABC99, BHP09, HPW09, FGKIS13, MPP^{+}14, MPS^{+}07, LMS12]).

Intuitivamente, la inconsistencia en Datalog$^{\pm}$ surge cuando una o más NCs o EGDs en una ontología son violadas, ya sea a través de la presencia de átomos en conflicto en el componente D o la generación de estos átomos a través de la aplicación de TGDs. Formalmente, la definición de inconsistencia en Datalog$^{\pm}$ se realiza siguiendo un punto de vista semántico, mediante la utilización de los modelos de una ontología KB [LMS12].

Definición 3.2 (Consistencia) *Una ontología Datalog*$^{\pm}$ $KB = (D, \Sigma)$ *es* consistente *si y sólo si* $\text{mods}(D, \Sigma) \neq \emptyset$.

Es decir, toda ontología inconsistente es tal que no tiene ningún modelo, lo que significa que no hay forma de satisfacer todas las restricciones de la misma al mismo tiempo (recordemos que un modelo es tal que toda fórmula en Σ es satisfecha).

Ejemplo 3.6 (Inconsistencia en ontologías Datalog$^{\pm}$) *Como un ejemplo de una ontología inconsistente, considere la mostrada a continuación.*

$$KB = \left\{ \begin{array}{rl} D\text{:} & \{a_1 : en\_terapia(charlie), a_2 : en\_relacion(kate, charlie), \\ & \; a_3 : terapista(kate), a_4 : pertenece\_a(g_1, charlie), \\ & \; a_5 : en\_terapia(patrick), a_6 : pertenece\_a(g_2, ed), \\ & \; a_7 : pertenece\_a(g_1, kate)\} \\ \\ \Sigma_{NC} : & \{\tau_1 : atendiendo(T, P) \wedge en\_relacion(T, P) \rightarrow \bot\} \\ \\ \Sigma_{E} : & \{\nu_1 : atendiendo(T, P) \wedge atendiendo(T', P) \rightarrow T = T'\} \\ \\ \Sigma_{T} : & \{\sigma_1 : en\_terapia(P) \rightarrow paciente(P), \\ & \; \sigma_2 : terapista(T) \wedge pertenece\_a(G, T) \rightarrow dirige(G, T), \\ & \; \sigma_3 : dirige(T, G), pertenece\_a(G, P) \rightarrow atendiendo(T, P), \\ & \; \sigma_4 : atendiendo(T, P) \rightarrow terapista(T)\} \end{array} \right\}$$

La ontología presentada es inconsistente. La instancia de base de datos D claramente no es un modelo en sí misma ya que al menos la TGD σ_2 es aplicable a D, pero no existe un superconjunto de D tal que el mismo satisfaga todas las TGDs, EGDs y NCs en Σ al mismo tiempo. Por ejemplo, la TGD σ_2 es aplicable para D creando como resultado el átomo dirige(kate, g_1) haciendo que σ_3 sea ahora aplicable, de cuya aplicación resulta el nuevo átomo atendiendo(kate, charlie), el cual junto con el átomo en_relacion(kate, charlie) (el cual ya se encontraba en D) viola la NC τ_1, ya que tenemos que un terapista esta involucrado sentimentalmente con un paciente suyo.

3.2.2. Incoherencia en Datalog$^{\pm}$

Hemos visto como problemas de consistencia pueden aparecer en Datalog$^{\pm}$. A continuación nos enfocaremos en otro problema diferente, pero a su vez relacionado a inconsistencia, la *incoherencia*. Muchos trabajos enfocados en el manejo de inconsistencia utilizan restricciones para enfocar la resolución de inconsistencias. Sin embargo, en general se asume que tales restricciones no tienen problemas y por lo tanto son satisfacibles. Un ejemplo de esto es el trabajo seminal de Konieczny y Pino-Pérez en integración de creencias [KP02] (donde se toman varias bases de conocimiento y se intenta integrarlas de manera consistente). Otros ejemplos son trabajos como [ABC99, LLR^{+}10, LMS12], donde se asume que el conjunto Σ expresa la semántica de los datos en la base de datos D, y por lo tanto se asume que no hay ningún conflicto interno en las restricciones; es decir, se asume que las restricciones son satisfacibles en el sentido de que su aplicación no provoca *inevitablemente* un problema de consistencia.

Sin embargo, tales suposiciones no siempre son aplicables. En la presente tesis elegimos considerar que tanto los datos como las restricciones pueden cambiar a medida que una ontología crece, y por lo tanto no sólo consideramos problemas relacionados a los datos sino también a las restricciones impuestas sobre los mismos. Nótese que en muchas aplicaciones del mundo real fórmulas en Σ puede traer aparejado problemas de consistencia. Por ejemplo, esto puede suceder cuando un conjunto de TGDs no puede ser aplicado sin llevar esto a una violación de una NC o una EGD.

El problema de *insatisfacibilidad de un concepto* en una ontología es conocido en la comunidad de DLs como *incoherencia* [FHP^{+}06, Bor95, BB97, KPSH05, SHCvH07, QH07]. En particular, en [FHP^{+}06] Flouris *et al.* establecen una relación entre inconsistencia e incoherencia, considerando esta última como una forma particular de la primera. En lugar de una teoría sin modelos, la noción de incoherencia que proponemos en la presente tesis dice que dado un conjunto de restricciones insatisfacibles Σ, no es posible encontrar un conjunto de átomos D que al mismo tiempo active todas las TGDs en Σ y un modelo exista para $KB = (D, \Sigma)$. Es decir, una KB puede ser consistente incluso si su conjunto de restricciones es incoherente, siempre y cuando la instancia de base de datos en la ontología no haga que las TGDs de la misma sean aplicables. Por el otro lado, una KB puede ser inconsistente incluso para una ontología coherente (y que por lo tanto tiene un conjunto satisfacible de restricciones); por ejemplo, considere $KB = (\{alto(pedro), bajo(pedro)\}, \{alto(X) \wedge bajo(X) \rightarrow \bot\})$, donde el conjunto (vacío)

de restricciones es trivialmente satisfacible haciendo la ontología coherente, pero sin embargo la ontología es inconsistente.

Definición de incoherencia

Antes de centrarnos en proveer una definición formal de incoherencia en ontologías Datalog$^{\pm}$, necesitamos identificar los conjuntos de átomos que son relevantes para un conjunto dado de restricciones. Intuitivamente, decimos que un conjunto de átomos A es relevante a un conjunto T de TGDs si los átomos en el conjunto A son tales que la aplicación de T sobre D genera los átomos que son necesarios para aplicar todas las TGDs en T. Presentamos a continuación la formalización de relevancia.

Definición 3.3 (Conjunto de átomos relevantes a un conjunto de TGDs) *Sea $\mathcal{R}$ un esquema relacional, $T \subseteq \Sigma_T$, y A un conjunto no vacío de átomos (posiblemente cerrado existencialmente), ambos sobre $\mathcal{R}$. Decimos que A es* relevante *a T si y sólo si para toda $\sigma \in T$ de la forma $\forall \mathbf{X} \forall \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z})$ se verifica que $chase(A, T) \models \exists \mathbf{X} \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$.*

En aquellos casos donde el contexto es suficiente para evitar confusión, cuando un conjunto unitario $A = \{a\}$ es relevante a $T \subseteq \Sigma_T$ directamente diremos que a es relevante a T.

Ejemplo 3.7 (Conjunto relevante de átomos) *Considere las siguientes TGDs:*

$$\Sigma_T = \{\sigma_1 : supervisa(X, Y) \rightarrow supervisor(X),$$
$$\sigma_2 : supervisor(X) \wedge toma\_desiciones(X) \rightarrow dirige\_departmento(X, D),$$
$$\sigma_3 : empleado(X) \rightarrow trabaja\_en(X, D)\}$$

Aquí, un ejemplo de conjunto relevante de átomos para el conjunto de TGDs $\Sigma_T = \{\sigma_1, \sigma_2, \sigma_3\}$ es el conjunto $A_1 = \{supervisa(walter, jesse), toma\_desiciones(walter), empleado(jesse)\}$, ya que σ_1 y σ_3 son directamente aplicables a A_1 y σ_2 se hace aplicable cuando aplicamos σ_1 (esto es, el chase genera el átomo $supervisor(walter)$, el cual junto con $toma\_decisiones(walter)$ dispara σ_2).

Sin embargo, $A_2 = \{supervisa(walter, jesse), toma\_desiciones(gus)\}$ *no es relevante a* Σ_T*. Nótese que incluso cuando* σ_1 *es aplicable a* A_2*, las TGDs* σ_2 *y* σ_3 *no son aplicables a* $chase(A_2, \Sigma_T)$*, ya que no podemos generar átomos que disparen tales TGDs. Por ejemplo, considere la TGD* $\sigma_2 \in \Sigma_T$*. En el chase de* Σ_T *sobre* D *generamos el átomo* $supervisor(walter)$*, pero de todas formas igual no podemos disparar* σ_2 *ya que no podemos generar el átomo* $toma\_decisiones(walter)$*, y el átomo* $toma\_decisiones(gus)$ *que se encuentra en* A_2 *no tiene el valor correcto de constante.*

A continuación definiremos el concepto de incoherencia en ontologías Datalog$^{\pm}$, adaptando la noción introducida para DLs por Flouris *et al.* en [FHP+06]. Nuestra concepción de (in)coherencia está basada en la insatisfacibilidad de un conjunto de TGDs. Intuitivamente, consideramos un conjunto de TGDs como satisfacible cuando existe un conjunto relevante de átomos que desencadena la aplicación de las TGDs en el conjunto y no produce la violación de ninguna restricción en $\Sigma_{NC} \cup \Sigma_E$, *esto es,* las TGDs pueden ser satisfechas junto con las NCs y las EGDs en *KB*.

Definición 3.4 (Satisfacibilidad) *Sea* $\mathcal{R}$ *un esquema relacional,* $T \subseteq \Sigma_T$ *un conjunto de TGDs, y* $N \subseteq \Sigma_{NC} \cup \Sigma_E$*, ambos sobre* $\mathcal{R}$*. El conjunto* T *es* satisfacible *respecto de* N *si y sólo si existe un conjunto* A *de átomos sobre* $\mathcal{R}$ *(posiblemente cerrado existencialmente) tal que* A *es relevante a* T *y* $\text{mods}(A, T \cup N) \neq \emptyset$*. Decimos que* T *es* insatisfacible *respecto de* N *si y sólo si* T *no es satisfacible respecto de* N*. Adicionalmente,* Σ_T *es satisfacible respecto de* $\Sigma_{NC} \cup \Sigma_E$ *si y sólo si no existe* $T \subseteq \Sigma_T$ *tal que* T *es insatisfacible respecto de algún* N *donde* $N \subseteq \Sigma_{NC} \cup \Sigma_E$*.*

En el resto de la disertación en ocasiones diremos que un conjunto de TGDs es (in)satisfacible omitiendo el conjunto de restricciones, hacemos esto en el contexto de una ontología particular donde tenemos un conjunto fijo de restricciones $\Sigma_{NC} \cup \Sigma_E$ ya que todo conjunto de TGDs que es satisfacible respecto de $\Sigma_{NC} \cup \Sigma_E$ es también satisfacible respecto de cualquier subconjunto del mismo y, por otra parte, cualquier conjunto insatisfacible de TGDs respecto de un subconjunto de $\Sigma_{NC} \cup \Sigma_E$ lo es también respecto del conjunto completo de restricciones.

Ejemplo 3.8 (Conjuntos satisfacibles e insatisfacible de TGDs) *Considere los siguientes conjuntos de restricciones.*

$$\Sigma^1_{NC} = \{\tau : \mathit{trabajo\_riesgoso}(P) \wedge \mathit{inestable}(P) \rightarrow \bot\}$$
$$\Sigma^1_{T} = \{\sigma_1 : \mathit{profesion\_peligrosa}(L) \wedge \mathit{trabaja\_en}(L, P) \rightarrow \mathit{trabajo\_riesgoso}(P),$$
$$\sigma_2 : \mathit{en\_terapia}(P) \rightarrow \mathit{inestable}(P)\}$$
$$\Sigma^1_{E} = \{\nu : \mathit{trabaja\_en}(L, P) \wedge \mathit{trabaja\_en}(L', P) \rightarrow L = L'\}$$

El conjunto Σ^1_T es un conjunto satisfacible de TGDS, y si bien la aplicación simultánea de σ_1 y σ_2 puede provocar la violación de una formula en $\Sigma^1_{NC} \cup \Sigma^1_E$, esto no se verifica para todo conjunto relevante de átomos. Considere por ejemplo el conjunto relevante $D_1 = \{profesion\_peligrosa(policia), trabaja\_en(policia, marty), en\_terapia(rust)\}$; D_1 es relevante para Σ^1_T, sin embargo, como tenemos que $\mathrm{mods}(D_1, \Sigma^1_T \cup \Sigma^1_{NC} \cup \Sigma^1_E) \neq \emptyset$ *entonces Σ^1_T es satisfacible.*

Por el otro lado, como un ejemplo de insatisfacibilidad (y por lo tanto incoherencia) considere el siguiente conjunto de restricciones, que expresa en Datalog$^{\pm}$, entre otras cosas, el conocimiento presentado en el Ejemplo 1.2 del Capítulo 1:

$$\Sigma^2_{NC} : \{\tau_1 : \mathit{garganta\_lastimada}(X) \wedge \mathit{puede\_cantar}(X) \rightarrow \bot,$$
$$\tau_2 : \mathit{tiene\_vida\_privada}(X) \wedge \mathit{famoso}(X) \rightarrow \bot\}$$
$$\Sigma^2_{E} : \{\nu_1 : \mathit{representa}(X, Y) \wedge \mathit{representa}(X, Z) \rightarrow Y = Z\}$$
$$\Sigma^2_{T} : \{\sigma_1 : \mathit{cantante\_rock}(X) \rightarrow \mathit{canta\_fuerte}(X),$$
$$\sigma_2 : \mathit{canta\_fuerte}(X) \rightarrow \mathit{garganta\_lastimada}(X),$$
$$\sigma_3 : \mathit{tiene\_fans}(X) \rightarrow \mathit{famoso}(X),$$
$$\sigma_4 : \mathit{cantante\_rock}(X) \rightarrow \mathit{puede\_cantar}(X),$$
$$\sigma_5 : \mathit{tiene\_fans}(X) \rightarrow \mathit{tiene\_vida\_privada}(X)\ \}$$

El conjunto Σ^2_T es un conjunto insatisfacible de dependencias, ya que (como explicamos en el Ejemplo 1.2) la aplicación de las TGDs $\{\sigma_1, \sigma_2, \sigma_3\}$ en cualquier conjunto relevante de átomos causará la violación de τ_1, v.g., si consideramos el átomo relevante $\mathit{cantante\_rock}(axl)$ tenemos que $Cn(\{\mathit{cantante\_rock}(axl)\}, \Sigma^2_T) = \{\mathit{cantante\_rock}(axl), \mathit{canta\_fuerte}(axl), \mathit{garganta\_lastimada}(axl), \mathit{puede\_cantar}(axl)\}$, y por lo tanto $\mathit{cantante\_rock}(axl)$ causa la violación de τ_1 cuando es considerado junto con Σ^2_T, lo que significa que $\mathrm{mods}(\{\mathit{cantante\_rock}(axl)\}, \Sigma^2_T \cup \Sigma^2_{NC} \cup \Sigma^2_E) = \emptyset$. *Nótese que* cualquier *conjunto relevante de átomos (esto es, cualquier cantante que consideremos) causará la violación de τ_1.*

Habiendo definido formalmente satisfacibilidad estamos en condiciones de definir coherencia para una ontología Datalog$^{\pm}$. Intuitivamente, una ontología es coherente si no existe un subconjunto de las TGDs que sea insatisfacible respecto del conjunto $\Sigma_{NC} \cup \Sigma_E$ en la ontología.

Definición 3.5 (Coherencia) *Sea $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ definida sobre un esquema relacional $\mathcal{R}$, y $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$, donde Σ_T es un conjunto de TGDs, Σ_E es un conjunto de EGDs separables y Σ_{NC} un conjunto de NCs. KB es* coherente *si y sólo Σ_T es satisfacible respecto de $\Sigma_{NC} \cup \Sigma_E$. Además, decimos que KB es incoherente si y sólo si no es coherente.*

Ejemplo 3.9 (Coherencia) *Considere los conjuntos de restricciones y dependencias definidos en el Ejemplo 3.8 y dos bases de datos arbitrarias D y D' definidas sobre los esquemas relacionales respectivos para cada conjunto de restricciones. Claramente, la ontología Datalog$^{\pm}$ $KB_1 = (D, \Sigma^1_T \cup \Sigma^1_{NC} \cup \Sigma^1_E)$ es coherente, mientras que $KB_2 = (D', \Sigma^2_T \cup \Sigma^2_{NC} \cup \Sigma^2_E)$ es incoherente.*

Relación entre incoherencia e inconsistencia

Algunos de los enfoques mas expandidos en el manejo de inconsistencia resuelven los conflictos mediante la remoción de átomos de la teoría. También es posible considerar simultáneamente todas las maneras de *reparar* una ontología mediante adición o remoción de átomos, como sucede en la mayoría de los enfoques para *Consistent Query Answering* [ABC99] (CQA).

Estos enfoques, sin embargo, pueden resultar inadecuados para el manejo de teorías incoherentes y producir resultados insignificantes; en casos extremos de incoherencia estos enfoques basados en datos pueden necesitar remover todos los átomos en la instancia de base de datos, produciendo una ontología sin ninguna información valiosa [DMFS15a]. Por esta razón, en la presente tesis consideramos que la incoherencia es importante en el proceso de consolidación de ontologías Datalog$^{\pm}$, ya que una de las propiedades esperadas de la ontología consolidada es la consistencia de la misma. Pero, como se dijo previamente, en el caso de un conjunto de restricciones incoherentes esto significa eliminar todo átomo relevante de D (en el peor escenario el conjunto D en su totalidad); lo cual no parece ser un resultado deseable en general.

Recapitulando las Definiciones 3.4 y 3.5 podemos ver que hay una estrecha relación entre los conceptos de incoherencia e inconsistencia. Es más, de las mismas puede inferirse que una *KB* incoherente inducirá una *KB* inconsistente cuando la instancia de base de datos contiene un conjunto de átomos que sea relevante a un conjunto insatisfacible de TGDs en Σ. Este resultado es capturado por la siguiente proposición.

Proposición 3.1 *Sea D un conjunto de átomos, $\Sigma_{_T}$ un conjunto de TGDs, $\Sigma_{_E}$ un conjunto de EGD separables y $\Sigma_{_{NC}}$ un conjunto de NCs. Sea $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ con $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$. Si KB es incoherente y existe $A \subseteq D$ tal que A es relevante a algún conjunto insatisfacible $U \subseteq \Sigma_{_T}$ entonces $KB = (D, \Sigma)$ es inconsistente.*

Demostración: ver Apéndice A [página 193].

Como una instancia particular de la relación mostrada en la Proposición 3.1 considere el siguiente ejemplo.

Ejemplo 3.10 (Relación entre incoherencia e inconsistencia) *Considere nuevamente los conjuntos de restricciones $\Sigma^2_{_T}$, $\Sigma^2_{_{NC}}$ y $\Sigma^2_{_E}$ del Ejemplo 3.8 tal que $\Sigma = \Sigma^2_{_T} \cup \Sigma^2_{_E} \cup \Sigma^2_{_{NC}}$, junto con la base de datos $D : \{a_1 : puede\_cantar(simone), a_2 : cantante\_rock(axl), a_3 : canta\_fuerte(ronnie), a_4 : tiene\_fans(ronnie), a_5 : cantante\_rock(ronnie), a_6 : cantante\_rock(roy), a_7 : representa(band_1, richard)\}$.*

Como hemos explicado brevemente en el Ejemplo 3.8, allí tenemos el conjunto $A \subset D = \{cantante\_rock(axl)\}$ y el conjunto insatisfacible de TGDs $U \subset \Sigma_{_T} = \{\sigma_1 : cantante\_rock(X) \rightarrow canta\_fuerte(X), \sigma_2 : canta\_fuerte(X) \rightarrow garganta\_lastimada(X), \sigma_4 : cantante\_rock(X) \rightarrow puede\_cantar(X)\}$. Como A es relevante a U se cumplen las condiciones de la Proposición 3.1, y efectivamente la ontología $KB = (D, \Sigma)$ es inconsistente ya que $\tau_1 \in \Sigma_{_{NC}}$ es violada.

Más adelante en la disertación analizaremos este ejemplo con más detalle; pero por el momento es suficiente con notar que en caso de querer hacer KB consistente mediante la remoción de átomos deberíamos eliminar todo aquel átomo relevante a un conjunto insatisfacible minimal.

3.3. Conclusiones

En los últimos tiempos han surgido entornos como la Web Semántica donde se hace preponderante el contar con contenidos procesables por computadoras. En tales entornos las ontologías y los sistemas basados en reglas han adquirido una gran importancia. Un foco importante de los desarrollos en tales áreas es la definición y el desarrollo de formalismos con alta capacidad de escalabilidad para ser utilizados en la Web de Datos (Web of Data), especialmente mediante generalizaciones de reglas de bases de datos y dependencias funcionales a través de axiomas ontológicos.

Uno de los desarrollos más importantes en tal sentido es el de la familia Datalog$^{\pm}$ de variantes de Datalog, los cuales extienden el Datalog básico con la posibilidad de utilizar cuantificaciones existenciales en las cabezas de las reglas, entre otras características importantes, manteniendo a su vez una tratabilidad adecuada para entornos de alto volumen de información mediante restricciones a la sintaxis de reglas. Tal familia de lenguajes se vuelve especialmente atrayente a los fines de esta tesis al considerar uno de los objetivos a largo plazo de la línea de investigación que anidó el desarrollo de la misma, el cual es la definición de una infraestructura que permita la realización de procesos de argumentación masiva sobre grandes volúmenes de datos almacenados en repositorios federados. Como veremos más adelante en el Capítulo 7, la expresividad de Datalog$^{\pm}$ nos brinda la posibilidad de trasladar los resultados de la presente tesis a otros formalismos que pueden ser representados por ontologías Datalog$^{\pm}$, como bases de datos relacionales, permitiendo la definición de repositorios federados a través de la utilización de operadores de consolidación.

La representación de conocimiento en Datalog$^{\pm}$ se hace a través de ontologías que combinan 4 diferentes componentes: un conjunto de átomos conocido como *base de datos*, un conjunto de reglas que permiten obtener nueva información a partir de las mismas conocidas como *tuple-generating dependencies*, restricciones sobre los átomos que pueden existir en el universo representadas a través de *negative constraints* y reglas que imponen la igualdad entre ciertos átomos conocidas como *equality-generating dependencies*. En base a tales componentes se puede obtener nueva información y resolver consultas. Para tal tarea se define un proceso que consiste en la aplicación sucesiva y exhaustiva de TGDs, un proceso conocido como *chase*, el cual puede ser utilizado para resolución de consultas en Datalog$^{\pm}$, ya que el chase de una ontología Datalog$^{\pm}$ es un *modelo universal* de la misma.

Finalmente, en el capítulo hemos abordado dos de los aspectos cruciales de la representación de conocimiento y el razonamiento, esto es inconsistencia e incoherencia, enfocándolos desde el punto de vista de Datalog$^{\pm}$. Con respecto a la primera, el hecho de usar restricciones como las EGDs y las NCs implica que en cierto momento las mismas sean violadas, llevando a tener ontologías inconsistentes. De esta forma, en la literatura de Datalog$^{\pm}$ ha sido señalado que las ontologías inconsistentes no poseen modelos, lo que tiene claramente una influencia directa en la resolución de consultas sobre las mismas, que se basa en los modelos (y en particular en el modelo universal obtenido a través del chase). De esta forma, el problema de inconsistencias en Datalog$^{\pm}$ debe ser abordado y resuelto de alguna forma. Como veremos en los siguientes capítulos de la presente disertación, los operadores aquí introducidos se enfocarán en la resolución de tales conflictos trabajando en conjuntos mínimos de átomos que impliquen una violación de consistencia en la ontología.

El otro concepto que suele tener gran importancia a la hora de expresar conocimiento es el de incoherencia, pero sin embargo hasta el momento ha sido dejado de lado por la literatura de Datalog$^{\pm}$, incluso cuando ha sido ampliamente reconocido en otros formalismos como Description Logics. En la presente tesis hemos formalizado tal concepto, asociándolo con la presencia de conjuntos de TGDs en una ontología tales que su activación (mediante su aplicación a cualquier conjunto de átomos que disparen las TGDs) lleva inevitablemente a violaciones de NCs o EGDs. Tales conjuntos de TGDs son, por lo tanto, insatisfacibles (al considerarlos junto con las restricciones expresadas a través de las NCs y las EGDs). Esto tiene un claro paralelo con los conceptos insatisfacibles en Description Logics, los cuales son tales que ningún individuo puede ser clasificado como representante del concepto; una ontología DL con tales conceptos es identificada como incoherente. Análogamente, en esta tesis decimos que una ontología Datalog$^{\pm}$ es incoherente cuando el conjunto de TGDs en la misma es insatisfacible con respecto a las restricciones en los conjuntos de NCs y EGDs en la ontología.

La importancia del manejo de incoherencias además de inconsistencias se pone en evidencia cuando nos detenemos a analizar la relación que se mantiene entre ambas: la activación de las reglas en cualquier conjunto insatisfacible de TGDs provoca inevitablemente una violación. De esta manera, si tenemos una ontología cuyo componente D es tal que el mismo activa un conjunto insatisfacible de TGDs entonces la ontología es necesariamente una ontología inconsistente. Por lo tanto, como veremos más adelante en

la presente tesis los operadores de consolidación de ontologías pueden beneficiarse de atacar no sólo problemas de inconsistencia sino también eliminando las incoherencias en el conocimiento expresado en las ontologías Datalog$^{\pm}$.

Capítulo 4

Revisión de Creencias

El principal objetivo de la teoría de cambio es tratar de modelar la dinámica del conocimiento, esto es, como éste cambia para adaptarse a influencias externas, *esto es,* después de recibir cierta información externa. El presente capítulo tiene como objetivo presentar los avances más importantes en la teoría, para de esta manera contribuir a hacer una tesis autocontenida; por lo tanto, en el mismo no se presentan nuevos aportes sino que se procede a establecer el marco formal donde los aportes que se presentarán en secciones subsecuentes han sido desarrollados. [1] En particular, primeramente presentamos una recapitulación de la historia del desarrollo de la teoría de cambio de creencias y de los principales operadores de cambio de creencias, a saber *expansión, contracción y revisión.* A su vez, extendemos tal recapitulación con una descripción de operadores que han tomado una mayor importancia en los últimos tiempos, en particular desde el advenimiento de entornos colaborativos; tales operaciones son las de *consolidación, combinación e integración de creencias.* Además, en el presente capítulo se presentan las relaciones entre los distintos operadores de cambio, incluyendo nuevas relaciones surgidas debido al surgimiento de nuevos operadores como los recientemente mencionados. Finalmente, siendo la presente tesis enfocada en el desarrollo sobre un lenguaje de representación de conocimiento preparado para entornos colaborativos como Datalog$^{\pm}$, presentaremos enfoques de cambio de creencias y manejo de inconsistencias en formalismos cercanos a Datalog$^{\pm}$ en términos de expresividad y adaptabilidad, como ser Description Logics y Logic Programming.

[1] Es importante aclarar que parte de lo presentado en este capítulo ha sido tomado y adaptado de las tesis doctorales del Dr. Marcelo Alejandro Falappa [Fal99] y el Dr. Luciano Héctor Tamargo [Tam10], en particular la presentación acerca de operadores de *expansión, contracción y revisión.*

4.1. Principales hitos en la historia de la revisión de creencias

Los orígenes de la teoría de revisión de creencias (*belief revision*) se remontan hacia los 1970s, cuando Isaac Levi discutió los problemas que conciernen a este campo de investigación [Lev77]. También, William Harper propuso una manera racional de relacionar algunos operadores de cambio de creencias [Har75]. Sin embargo, el principal avance en la teoría de cambio de creencias fue durante los 1980s cuando Carlos Alchourrón y David Makinson estudiaron los cambios en códigos legales [AM81] y Peter Gärdenfors abordaba los postulados racionales para operadores de cambio [Gär82]. Este avance quedó marcado después de que los tres autores escribieran un artículo internacionalmente conocido como modelo AGM [AGM85], llamado así por las iniciales de los apellidos de sus tres creadores. Dicho modelo proveyó el marco de trabajo formal más general hasta ese momento para estudios de cambio de creencias y representa un punto de quiebre en la evolución de la teoría de cambio de creencias.

Hay muchos marcos de trabajos diferentes para revisión de creencias con sus respectivos modelos epistémicos. El modelo epistémico es el formalismo en el cual las creencias son representadas y en el cual pueden definirse diferentes clases de operadores. El modelo AGM representa los estados epistémicos por medio de conjuntos de creencias (conjuntos de sentencias cerrados bajo consecuencia lógica) y la entrada epistémica por una única sentencia. AGM define tres operadores de cambio básicos: expansión, contracción y revisión. Las contracciones en AGM son llamadas *partial meet contractions* y sus operadores de revisión asociados (mediante la identidad de Levi) son llamados *partial meet revisions*. Ambas clases de operadores pueden ser presentados en dos maneras: dando una construcción explícita (*v.g.,* a través de algoritmos) para el operador, o dando un conjunto de postulados de racionalidad a ser satisfechos. Los postulados de racionalidad [AGM85, Gär88] determinan restricciones que los operadores deben satisfacer. Ellos tratan a los operadores como cajas negras, describiendo sus comportamientos con respecto a la entrada en casos básicos, pero no los mecanismos internos usados. El impacto de AGM en la teoría de cambio de creencias fue enorme; no sólo se trató de uno de los primeros y más importantes desarrollos en cuanto a operadores de cambio de creencias en sí, sino que también el trabajo de Alchourrón, Makinson y Gärdenfors sentó las bases de un "estilo" en la presentación de resultados en el área, mediante la utilización de los

teoremas de representación provistos (también llamados caracterizaciones axiomáticas) de la contracción y la revisión, los cuales sirven para brindar una clara conexión entre las construcciones de los operadores y las propiedades esperadas de los mismos.

Sin dudas, por lo expuesto anteriormente podemos afirmar que el modelo AGM ha sido el trabajo más influyente en la teoría de cambio de creencias. Sin embargo, hay modelos equivalentes tales como: *safe contractions* [AM85], *epistemic entrenchment* [GM88], *sphere systems* [Gro88], y *kernel contractions* [Han94]. Supongamos que K representa el conocimiento actual y α representa una nueva pieza de información. Supongamos que una contracción de K por α es realizada. Siguiendo el modelo AGM, una *partial meet contraction* está basada en una selección entre los subconjuntos maximales de K que no implican α. Otro posible enfoque para las contracciones está basado en una selección entre los subconjuntos minimales de K que contribuyen a implicar α tal como en safe contraction [AM85]. Una variante más general del mismo enfoque fue introducido más tarde y es conocido como *kernel contraction* [Han94].

Una tercera construcción sigue el concepto de *Epistemic Entrenchment*. De acuerdo a [Gär03]: "*Ciertas piezas de nuestro conocimiento y creencias acerca del mundo son más importantes que otras cuando se planean acciones futuras, conduciendo investigaciones científicas, o razonando en general. Diremos que algunas sentencias en un sistema de creencias tienen más alto grado de importancia epistémica (epistemic entrenchment) que otras....* ". Epistemic entrenchment permite que una contracción (o revisión) pueda estar basada en una relación entre sentencias que, cuando es forzada a elegir entre dos creencias, un agente descartará la menos importante. Además, ha sido mostrado que una contracción basada en una relación de epistemic entrenchment es equivalente a una *partial meet contraction* [GM88].

El sistema de esferas de Grove [Gro88] es muy similar a la semántica de esferas para contrafácticas propuesto por David Lewis [Lew73]. El sistema de Grove son esferas concéntricas que permiten el ordenamiento de mundos posibles de acuerdo a alguna noción de similitud o preferencia. Luego, el resultado de revisar por α puede ser semánticamente descripto por el conjunto de α-modelos que son los más cercanos a los K-modelos. Se ha mostrado que estas visiones usando el sistema de esferas de Grove es equivalente a una *partial meet revision* [Gro88].

Otro de los modelos propuestos es *updating* [KM91]. Mientras que en revisión viejas creencias y la nueva información se refieren a la misma situación, en *updating* la nueva

información es acerca de una situación actual. La meta principal de revisión de creencias es cambiar las creencias preservando tanto como sea posible la vieja información; el objetivo del *updating* es sólo cambiar los mundos en los cuales el cambio es posible. Todos ellos son modelos priorizados, de manera que cuando un cambio es realizado, la nueva información tiene primacía sobre las creencias del conjunto original. Una presentación completa de los modelos clásicos priorizados de revisión de creencias pueden ser encontrados en [Han99], y un conjunto amplio y general para revisiones no priorizadas puede ser encontrado en [Han97a, HFCF01].

4.2. Modelo epistémico y estado epistémico

Como hemos mencionado anteriormente, un modelo epistémico es el formalismo en el cual las creencias o el conocimiento son representadas y en el cual diferentes clases de operadores pueden ser definidos. En cambio, un estado epistémico o estado de creencias es una representación del conocimiento y creencias en un momento en el tiempo [Gär88]. Es claro que este concepto está fuertemente conectado con la teoría de cambio de creencias. Los estados epistémicos no son entidades psicológicas; son presentadas como idealizaciones racionales de estados psicológicos. Esto es, computacionalmente un estado epistémico puede estar representado físicamente mediante una secuencia de 0's y 1's. Los estados epistémicos son las entidades centrales en las teorías epistemológicas y se usan para representar el posible estado cognitivo de un individuo en un momento del tiempo. El concepto de un estado epistémico no se entiende como un concepto que expresa algo acerca de como el conocimiento y las creencias se representan y son tratadas en nuestro cerebro; sino que es un concepto epistemológico que se entiende como una idealización del concepto psicológico [Gär88].

Si bien en la presente tesis nos enfocaremos en un lenguaje particular como Datalog$^{\pm}$ para la definición de procesos de cambio (en particular, consolidación de creencias), en el presente capítulo, por razones de simplicidad, y a fin de introducir los conceptos básicos de la teoría de cambio de creencias, nos enfocaremos en un lenguaje proposicional $\mathcal{L}$. También asumimos la existencia de un operador Cn que incluye el operador de consecuencia clásica, y que para todo conjunto de creencias B satisface las siguientes propiedades: (a) *inclusión*: $B \subseteq Cn(B)$, (b) *iteración*: $Cn(B) = Cn(Cn(B))$, (c) *monotonicidad*: si $B \subseteq C$ luego $Cn(B) \subseteq Cn(C)$, y (d) *compacidad*: si $\alpha \in Cn(B)$, entonces $\alpha \in Cn(B')$ para algún

subconjunto finito $B' \subseteq B$. En general, en la presente sección escribiremos $\alpha \in Cn(B)$ como $B \vdash \alpha$.

Conjuntos de creencias

El objetivo de estudiar las operaciones de cambio sobre conjuntos de creencias (*esto es,* sobre conjuntos de sentencias de un lenguaje determinado clausuradas bajo cierta operación de consecuencia lógica) permite reconocer más directamente las propiedades de las mismas sin perder de vista las intuiciones que las motivaron. Este es conocido como el *modelo de coherencia* dentro de la literatura de teoría de cambio de creencias. Si bien no es posible tratar con conjuntos de creencias en una computadora (puesto que en general, los mismos son infinitos) es posible caracterizar las propiedades que debe satisfacer cada una de las operaciones de cambio sobre representaciones finitas de un estado de creencias. En tal sentido, Mukesh Dalal [Dal88] ha formulado el principio de irrelevancia de la sintaxis para bases de datos (o bases de creencias), el cual establece que el resultado de una operación de cambio sobre bases de creencias, bases de conocimiento o bases de datos no deberían depender de la sintaxis (representación) tanto en la información original como en la información actualizada. Esto significa que el resultado de una operación de cambio no debería depender de la base de creencias usada para representar un estado epistémico. Siguiendo la tradición en la literatura de la teoría de cambio de creencias desde este punto en adelante denotaremos a los conjuntos de creencias utilizando letras en negrita, *v.g.,* $\mathbf{K}$.

Bases de creencias

Las bases de creencias se representan mediante un conjunto de sentencias no necesariamente clausurado, lo que es conocido como *modelo fundacional*. Esta característica hace que las operaciones de cambio sobre conocimiento representado mediante bases de creencias sean computacionalmente tratables. Esto puede hacernos pensar que nuestro esfuerzo en el desarrollo de operaciones de cambio debe estar centrado en bases de creencias y no en conjuntos de creencias. Sin embargo, veremos que esta primera visión parece un poco apresurada. Los modelos que emplean bases de creencias están fundamentados en la intuición de que algunas de nuestras creencias no tienen una sustentación independiente, sino que surgen como consecuencia de aplicar reglas de inferencia sobre nuestras creencias más fundamentales, de las cuales dependen totalmente [GR92, HR95].

En resumen, el uso de bases de creencias hace la representación del conocimiento más natural y computacionalmente tratable. Esto es, siguiendo [Han99, página 24], consideramos que el conocimiento esta expresado a través de un número limitado de sentencias que corresponden a las creencias explícitas en el mismo. Nuevamente, continuando con lo tradicionalmente establecido en cambio de creencias en la presente tesis las bases de creencias serán denotadas utilizando letras en itálica, *v.g.,* K.

Un punto importante a notar es que toda operación de cambio sobre una base de creencias tiene asociada una operación de cambio sobre un conjunto de creencias (que se obtiene clausurando la base de creencias). Esto es, K es una base de creencias para un conjunto de creencias $\mathbf{K}$ si su clausura lógica coincide con el conjunto de creencias $\mathbf{K}$. Luego, podemos definir un operador de cambio sobre el conjunto de creencias $\mathbf{K}$ a partir de uno más elemental definido sobre la base (K) que lo genera.

4.3. Operaciones básicas de cambio en el conocimiento

A continuación introducimos las operaciones clásicas dentro de la Teoría de Cambio; a saber expansión, contracción y revisión.

4.3.1. Expansión

En esta sección trataremos la más simple de las operaciones de cambio. Tal operación se denomina *expansión* y consiste en el agregado de nueva información al estado epistémico de un agente. La definición de la misma puede darse en términos puramente lógicos. Por tal motivo, iremos dando una a una, las propiedades que debe satisfacer todo operador de expansión sobre conjuntos de creencias hasta llegar a una definición concreta del proceso de cambio.

Propiedades de un operador de expansión

Las operaciones de expansión toman lugar cuando se observan cambios o existe información externa provista por otros agentes (por ejemplo, personas) mediante algún canal

de comunicación. Para simplificar la caracterización de un operador de expansión, asumiremos que $\mathbf{K}$ es un *conjunto de creencias*, es decir, $\mathbf{K}$ es cerrado bajo consecuencia lógica. El operador de expansión será representado mediante el símbolo +, el cual es una función que, dado un conjunto de creencias y una sentencia del lenguaje relaciona un nuevo conjunto de creencias. A continuación daremos los *Postulados de Racionalidad para Expansiones* que fueron introducidos por Peter Gärdenfors para caracterizar las operaciones de cambio [Gär88].

Partiendo del hecho de que el estado de creencias se representa mediante un conjunto de creencias $\mathbf{K}$ el cual incluye todas sus consecuencias lógicas (*esto es,* $\mathbf{K} = Cn(\mathbf{K})$), es deseable que el conjunto expandido sea un conjunto de creencias, esto es, $\mathbf{K}{+}\alpha = Cn(\mathbf{K}{+}\alpha)$. Esto da lugar al primer postulado para expansiones:

(E^+1) *Clausura* (Closure): Para cualquier conjunto de creencias $\mathbf{K}$ y cualquier sentencia α, $\mathbf{K}{+}\alpha$ es un conjunto de creencias.

Si expandimos un conjunto de creencias $\mathbf{K}$ con respecto a una fórmula α es natural que α sea aceptada en el conjunto de creencias expandido. Esta idea da lugar al segundo postulado para expansiones:

(E^+2) *Éxito* (Success): $\alpha \in \mathbf{K}{+}\alpha$.

Puesto que obtener información es un proceso muy costoso, es deseable evitar pérdida innecesaria de información en cualquier operación de cambio. Este criterio heurístico es llamado criterio de *economía informacional* y juega un rol importante en la teoría de cambio de creencias (particularmente en el modelo AGM). Si $\mathbf{K}$ es consistente y α está en $\mathbf{K}$ entonces $\neg\alpha$ no está en $\mathbf{K}$. En ese caso, α no contradice las creencias de K. Por lo tanto, si $\neg\alpha \notin \mathbf{K}$, podemos retener tanto como sea posible de las creencias originales y respetamos el criterio de economía informacional:

$$\text{Si } \neg\alpha \notin \mathbf{K}, \text{ entonces } \mathbf{K} \subseteq \mathbf{K}{+}\alpha \tag{4.1}$$

Sin embargo, en el caso en que $\neg\alpha$ pertenezca a $\mathbf{K}$, la expansión de $\mathbf{K}$ por α produce una inconsistencia. Luego, como el conjunto de creencias expandido está clausurado bajo consecuencia lógica entonces $\mathbf{K}{+}\alpha$ es el *conjunto de creencias absurdo* el cual lo notaremos $\mathbf{K}_\perp$ y es igual a $\mathcal{L}$.

$$\text{Si } \neg\alpha \in \mathbf{K}, \text{ entonces } \mathbf{K}{+}\alpha = \mathbf{K}_\perp \tag{4.2}$$

Como $H \subseteq \mathbf{K}_{\perp}$ para todo $H \subseteq \mathcal{L}$ entonces, a partir de las fórmulas 4.1 y 4.2 podemos obtener el tercer postulado para expansiones el cual motiva el nombre de la operación de cambio:

(E^+3) *Inclusión*: $\mathbf{K} \subseteq \mathbf{K}{+}\alpha$.

Un caso especial de expansión ocurre cuando expandimos por una creencia en la que ya creemos. Esto es, cuando expandimos $\mathbf{K}$ por α y α está en $\mathbf{K}$. En ese caso, la entrada epistémica no produce cambios en el conjunto de creencias expandido, puesto que α ya está en $\mathbf{K}$. Esta propiedad da lugar al cuarto postulado para expansiones:

(E^+4) *Vacuidad* (Vacuity): Si $\alpha \in \mathbf{K}$ entonces $\mathbf{K}{+}\alpha = \mathbf{K}$.

Supongamos contar con dos conjuntos de creencias, uno de los cuales está contenido en el otro. Esto es, contamos con dos estados epistémicos de los cuales uno de ellos está "más informado" que el otro. Si expandimos ambos estados con respecto a una misma creencia es deseable que la relación de inclusión entre los estados epistémicos expandidos se preserve. Esta propiedad da lugar al quinto postulado para expansiones:

(E^+5) *Monotonía*: Si $H \subseteq \mathbf{K}$ entonces $H{+}\alpha \subseteq \mathbf{K}{+}\alpha$.

Los postulados E^+1...E^+5 no excluyen la posibilidad de que $\mathbf{K}{+}\alpha$ contenga creencias no pertenecientes a $\mathbf{K}$ que no tengan ninguna conexión con α (excepto en el caso límite que $\alpha \in \mathbf{K}$ puesto que en tales circunstancias $\mathbf{K}{+}\alpha = \mathbf{K}$). Es deseable que $\mathbf{K}{+}\alpha$ no contenga más creencias que las requeridas por los otros postulados. Esto motiva el sexto y último postulado para expansiones:

Para todo conjunto de creencias $\mathbf{K}$ y toda sentencia α, $\mathbf{K}{+}\alpha$ es el más pequeño conjunto de creencias que satisface $E^+1 \ldots E^+5$.

Este postulado puede escribirse también de la siguiente forma:

(E^+6) $\mathbf{K}{+}\alpha \subseteq Cn(\mathbf{K} \cup \{\alpha\})$.

Los postulados E^+1...E^+5 aseguran que se satisfaga el siguiente requerimiento:

$$Cn(\mathbf{K} \cup \{\alpha\}) \subseteq \mathbf{K}{+}\alpha$$

El postulado E^+6 permite transformar la inclusión anterior en una igualdad. Esto da lugar al siguiente *teorema de representación* el cual da una definición explícita del proceso de expansión.

Teorema 4.1 *(AGM [AGM85]) La operación de expansión "+" satisface* $E^+1 ... E^+6$ *si y solo si:*

$$\mathbf{K}+\alpha = Cn(\mathbf{K} \cup \{\alpha\})$$

4.3.2. Contracción

Una contracción pura [Han99] consiste en una operación de cambio en la cual se eliminan creencias sin el agregado de ninguna creencia nueva. Una operación de contracción toma lugar cuando un agente constata que ciertas creencias que componen su conocimiento dejan de ser ciertas, o bien, como consecuencia de la descomposición de otra operación de cambio.

A continuación mostraremos dos operadores de contracción de manera constructiva y por medio de teoremas de representación. Primero introduciremos el operador de *partial meet contraction* presentado en el modelo AGM [AGM85] para conjuntos de sentencias y luego mostraremos el operador de *kernel contraction* definido en [Han94].

Partial meet contractions

Las funciones de contracción *partial meet contraction* fueron definidas por Alchourrón, Gärdenfors y Makinson [AGM85] y constituyen una de las clases más generales dentro de las operaciones de contracción puesto que la mayoría de las contracciones se definen como un tipo especial de *partial meet contraction*.

Construcción

Supongamos que se desea contraer un conjunto de creencias $\mathbf{K}$ con respecto a una creencia α. Una operación de *partial meet contraction* consiste en lo siguiente:

1. Se obtienen los subconjuntos maximales de $\mathbf{K}$ que fallan en implicar α.
2. Se seleccionan los "mejores" subconjuntos con esas características.
3. Se intersectan esos subconjuntos seleccionados y se obtiene la función de contracción correspondiente.

Con el fin de expresar formalmente estas ideas, definiremos la noción de *conjunto de restos* o *conjunto restante* (*remainder set*).

Definición 4.1 *AGM [AGM85] Sean K un conjunto de sentencias y α una sentencia. El conjunto $K^{\perp}\alpha$, denominado conjunto de restos, es el conjunto de conjuntos X tales que:*

1. $X \subseteq K$.
2. $\alpha \notin Cn(X)$.
3. *No existe Y tal que $X \subset Y \subseteq K$ tal que $\alpha \notin Cn(Y)$.*

Ya hemos definido el conjunto de restos. Ahora definiremos formalmente el concepto de función de selección. Esta función se encargará de seleccionar los "mejores elementos" del conjunto de restos.

Definición 4.2 *AGM [AGM85] Sean K un conjunto de sentencias, α una sentencia y $K^{\perp}\alpha$ el conjunto de restos de K con respecto a α. Decimos que "γ" es una función de selección si y solo si:*

1. *Si $K^{\perp}\alpha = \emptyset$ entonces $\gamma(K^{\perp}\alpha) = \{K\}$.*
2. *Si $K^{\perp}\alpha \neq \emptyset$ entonces $\gamma(K^{\perp}\alpha) = X$ tal que $\emptyset \subset X \subseteq (K^{\perp}\alpha)$.*

En realidad esta definición no determina unívocamente una función sino que define una *clase* o *familia* de funciones con ciertas características. A partir del conjunto de restos y la función de selección, podemos obtener la primera definición constructiva de las operaciones de *partial meet contraction*.

Definición 4.3 *AGM [AGM85] Sean $\mathbf{K}$ un conjunto de creencias, α una sentencia y $\mathbf{K}^{\perp}\alpha$ el conjunto de restos de $\mathbf{K}$ con respecto a α. Sea "γ" una función de selección para $\mathbf{K}^{\perp}\alpha$. El operador "$-_{\gamma}$", denominado partial meet contraction determinado por "γ" se define de la siguiente manera:*

$$\mathbf{K}-_{\gamma}\alpha = \bigcap \gamma(\mathbf{K}^{\perp}\alpha)$$

La idea intuitiva de esto es que la función de selección "γ" elija los subconjuntos maximales de $\mathbf{K}^{\perp}\alpha$ que sean más importantes en algún sentido epistemológico.

En la definición de *partial meet contraction*, la función de selección "γ" se asume como dada. Pero si deseamos obtener un método de contracción más específico, debemos saber como determinar los elementos a seleccionar basándonos en un criterio de preferencia epistemológica.

Propiedades

En la presentación de las propiedades utilizaremos un *conjunto de creencias* $\mathbf{K}$ arbitrario. El operador de contracción será representado mediante el símbolo "$-$", el cual es una función que, dado un conjunto de creencias y una sentencia del lenguaje, produce un nuevo conjunto de creencias.

A continuación daremos los *Postulados de Racionalidad para Contracciones* [Gär88]. Estos postulados no son suficientes para determinar unívocamente una operación de contracción ya que existen otros factores epistémicos que juegan un rol importante. Sin embargo, tales postulados permiten dar una caracterización general, y por lo tanto incompleta, de una operación de contracción.

El primer postulado para contracciones establece que si contraemos un conjunto de creencias con respecto a una creencia determinada, obtenemos un nuevo conjunto de creencias (cerrado bajo consecuencia lógica), esto es, $\mathbf{K}{-}\alpha = Cn(\mathbf{K}{-}\alpha)$. Más formalmente esto equivale a decir:

(C-1) *Clausura* (Closure): Para cualquier conjunto de creencias $\mathbf{K}$ y cualquier sentencia α, $\mathbf{K}{-}\alpha$ es un conjunto de creencias.

Puesto que $\mathbf{K}{-}\alpha$ surge a partir de la eliminación de ciertas creencias de $\mathbf{K}$ sin el agregado de ninguna creencia, es de esperar que $\mathbf{K}{-}\alpha$ no contenga ninguna creencia que no pertenece a $\mathbf{K}$. Esto da lugar al segundo postulados para contracciones:

(C-2) *Inclusión* (Inclusion): $\mathbf{K}{-}\alpha \subseteq \mathbf{K}$

Cuando $\alpha \notin \mathbf{K}$, el criterio de economía informacional exige que ninguna creencia sea eliminada de $\mathbf{K}{-}\alpha$. Este tercer postulado para contracciones está relacionado a la idea de mínimo cambio.

(C-3) *Vacuidad* (Vacuity): Si $\alpha \notin \mathbf{K}$ entonces $\mathbf{K}{-}\alpha = \mathbf{K}$.

Cuando se realiza una contracción de un conjunto de creencias $\mathbf{K}$ con respecto a una creencia α, es deseable que α no pertenezca al conjunto de creencias contraído, salvo que α sea un teorema o una sentencia lógicamente válida. Este es el cuarto postulado para contracciones:

(C-4) *Éxito* (Success): Si $\nvdash \alpha$ entonces $\alpha \notin \mathbf{K}{-}\alpha$.

Si α fuera un teorema[2], entonces $\alpha \in Cn(\emptyset)$. Como $\emptyset \subseteq H$ para todo $H \in \mathcal{K}_{\mathcal{L}}$, por propiedad de monotonía del operador de consecuencia Cn, tenemos que $Cn(\emptyset) \subseteq Cn(H)$. Luego, $\alpha \in H$ para todo $H \in \mathcal{K}_{\mathcal{L}}$. Esto es, $\alpha \in Cn(\emptyset)$ debe pertenecer a las consecuencias de cualquier conjunto de creencias (contraído o no).

A partir de los postulados (C-1)...(C-4) y de los postulados para expansiones $(E^{+}1)...(E^{+}6)$ es posible obtener la siguiente propiedad:

$$\text{Si } \alpha \in \mathbf{K}, \text{ entonces } (\mathbf{K}{-}\alpha){+}\alpha \subseteq \mathbf{K}$$

Esta propiedad garantiza que al contraer un conjunto de creencias con respecto a α y luego expandir el conjunto de creencias resultante con respecto a la misma sentencia, el conjunto final no contendrá ninguna creencia que no estaba en el conjunto de creencias original. Sin embargo, el criterio de economía informacional exige que $\mathbf{K}{-}\alpha$ sea el mayor subconjunto posible de $\mathbf{K}$. En analogía con (E+6), el siguiente postulado [AGM85, Gär88] exige que *todas* las creencias en $\mathbf{K}$ deben *recuperarse* como consecuencia de contraer e inmediatamente expandir un conjunto de creencias con respecto a una creencia cualquiera. Este postulado, conocido como *recuperación* (*recovery*) [AGM85, Gär88] es uno de los más controvertidos de toda la teoría de cambio de creencias (principalmente en el modelo AGM) y ha motivado un gran debate en la comunidad científica. Formalmente, tal postulado puede expresarse de la siguiente forma:

$$\text{Si } \alpha \in \mathbf{K} \text{ entonces } \mathbf{K} \subseteq (\mathbf{K}{-}\alpha){+}\alpha.$$

Si $\alpha \notin \mathbf{K}$ entonces (por vacuidad) $\mathbf{K}{-}\alpha = \mathbf{K}$. Esto implica que $\mathbf{K} \subseteq (\mathbf{K}{-}\alpha){+}\alpha = \mathbf{K}{+}\alpha$. Por lo tanto, el postulado de recuperación también puede definirse como sigue:

(C-5) *Recuperación* (Recovery): $\mathbf{K} \subseteq (\mathbf{K}{-}\alpha){+}\alpha$.

[2]Es importante remarcar que todas las sentencias demostrables en una teoría son teoremas de la misma. Sin embargo, en el presente capítulo cuando hablemos de teorema nos estaremos refiriendo a aquellas sentencias demostrables en la teoría que pueden demostrarse usando lógica proposicional.

El postulado de recuperación puede valer cuando se realizan operaciones de cambio sobre conjuntos de creencias pero no ocurre lo mismo cuando se realiza sobre bases de creencias. Cuando tratamos con bases de creencias, si deseamos contraer K con respecto a α, debemos eliminar α o bien otras sentencias con el objetivo de eliminar cualquier posible derivación de la misma. Sin embargo, como K no está clausurada bajo consecuencia lógica, una vez eliminadas las creencias que permiten la deducción de α pero que no se deducen a partir de α, es imposible poder recuperarlas con una expansión (con respecto a α) de la base de creencias contraída.

En la operación de contracción es deseable que se respete el principio de irrelevancia de la sintaxis, esto es, la contracción de un conjunto de creencias con respecto a sentencias lógicamente equivalentes debe producir el mismo resultado. Este postulado se conoce como *extensionalidad* (*extensionality*) o *preservación* (*preservation*), y se define como sigue:

(C-6) *Extensionalidad/Preservación*: Si $\vdash \alpha \leftrightarrow \beta$ entonces $\mathbf{K}-\alpha = \mathbf{K}-\beta$.

Los operadores de contracción del tipo *partial meet* satisfacen los postulados básicos para contracciones. Esto da lugar al siguiente teorema.

Teorema 4.2 *AGM [AGM85] Sea "−" un operador de contracción. Diremos que "−" es un operador de* partial meet contraction *si y solo si satisface* $(C-1)...(C-6)$*, es decir, satisface clausura, inclusión, vacuidad, éxito, recuperación y extensionalidad.*

Las siguientes propiedades también permiten obtener otra caracterización axiomática de las operaciones de *partial meet contraction* sobre conjuntos arbitrarios (no necesariamente clausurados)[3].

Uniformidad (Uniformity) (Hansson [Han92]) Para todo subconjunto K' de K vale que si $\alpha \in Cn(K')$ si y solo si $\beta \in Cn(K')$ entonces $K-\alpha = K-\beta$.

Esta propiedad establece que si dos sentencias α y β son implicadas exactamente por los mismos subconjuntos de K entonces la contracción de K por α debería ser igual a la contracción de K por β.

Relevancia (Relevance) (Hansson [Han89, Han92]) Si $\beta \in K$ y $\beta \notin K-\alpha$ entonces existe un conjunto K' tal que $K-\alpha \subseteq K' \subseteq K$, $\alpha \notin Cn(K')$ pero $\alpha \in Cn(K' \cup \{\beta\})$.

[3]Si bien en su presentación utilizaremos la notación K, las propiedades expresadas por Hansson son válidas tanto para conjuntos de creencias como para bases de creencias. Es decir, las propiedades valen para todo conjunto de sentencias, clausurado o no.

Retención de Núcleo (Core Retainment) (Hansson [Han91b]) Si $\beta \in K$ y $\beta \notin K-\alpha$ entonces existe un conjunto K' tal que $K' \subseteq K$, $\alpha \notin Cn(K')$ pero $\alpha \in Cn(K' \cup \{\beta\})$.

La propiedad de retención de núcleo determina que si una sentencia β es excluida en la contracción de K por α, entonces β contribuye de alguna manera al hecho de que K implique a α [Han99]. Relevancia, por su parte, determina que β contribuye al hecho de que K (pero no $K-\alpha$) implique a α. Claramente, el postulado de retención de núcleo es más débil (o menos restrictivo) que el postulado de relevancia.

El siguiente teorema da otra caracterización axiomática de las operaciones de *partial meet contraction* sobre conjuntos no necesariamente clausurados.

Teorema 4.3 *Hansson [Han92] Sea "$-$" un operador de contracción. Diremos que "$-$" es un operador de* partial meet contraction *para un conjunto K si y solo si satisface* inclusión, éxito, relevancia *y* uniformidad.

Kernel contractions

Las operaciones de *partial meet contraction* se basan en la selección de subconjuntos maximales de K que fallan en implicar α. Para hacer posible esto es necesario contar con una relación de preferencia entre los subconjuntos del conjunto K. Si K es cerrado bajo consecuencia lógica entonces es de una cardinalidad infinita por lo que la factibilidad de "encontrar" una operación de *partial meet contraction* sobre un conjunto de creencias resulta poco tratable. Otra propuesta diferente, que es aplicable tanto a bases de creencias como a conjuntos de creencias, consiste en construir un operador de contracción que selecciona entre los elementos de K que contribuyen a implicar α aquellos que serán descartados. Este tipo de operación de contracción fue inicialmente introducida por Alchourrón y Makinson [AM85] y se conoce como *safe contraction*. Una variante más general de esta propuesta es la que se conoce como *Kernel Contraction* [Han94] que fue introducida por Hansson un tiempo después.

Construcción

Las sentencias que se eliminan en una operación de *Kernel Contraction* de un conjunto K con respecto a una creencia α son aquellas creencias que de alguna manera contribuyen

a que K derive α. Para poder definir la operación de *Kernel Contraction*, daremos una nueva definición de conjunto de kernels.

Definición 4.4 *Hansson [Han94] Sean K un conjunto de sentencias y α una sentencia. El conjunto $K^{\perp\!\!\!\perp}\alpha$, denominado conjunto de Kernels es el conjunto de conjuntos K' tales que:*

1. $K' \subseteq K$.
2. $K' \vdash \alpha$.
3. *Si* $K'' \subset K'$ *entonces* $K'' \nvdash \alpha$.

El conjunto $K^{\perp\!\!\!\perp}\alpha$ también se denomina conjunto de α-kernels y cada uno de sus elementos se denomina α-kernel. Cada kernel o α-kernel también es conocido como (entailment set) [Fuh91].

Ejemplo 4.1 *Sea K el conjunto formado por las siguientes sentencias:*

$$\{a, b, a \to b, b \to a, c\}$$

donde a, b y c son sentencias lógicamente independientes. Supongamos que deseamos contraer con respecto a la fórmula $a \wedge b$. El conjunto de $(a \wedge b)$-kernels es el siguiente:

$$K^{\perp\!\!\!\perp}(a \wedge b) = \{\{a, b\}, \{a, a \to b\}, \{b, b \to a\}\}$$

Cada uno de los elementos de $K^{\perp\!\!\!\perp}(a \wedge b)$ implica la fórmula $(a \wedge b)$. A su vez, cada uno de estos conjuntos es minimal en el sentido de que no existe ningún subconjunto estricto de ellos que derive $(a \wedge b)$.

Para que la operación de contracción de K con respecto a $(a \wedge b)$ sea exitosa es necesario eliminar al menos un elemento de cada $(a \wedge b)$-kernel. Haciendo esto, se garantiza que la operación de contracción satisfaga la propiedad de éxito. Los elementos a ser eliminados son seleccionados por una *función de incisión* la cual es denominada así porque realiza un "corte" sobre cada kernel.

Definición 4.5 *Hansson [Han94] Una función "σ" es una función de incisión para K si para toda sentencia α vale que:*

1. $\sigma(K^{\perp\!\!\!\perp}\alpha) \subseteq \cup(K^{\perp\!\!\!\perp}\alpha)$.

2. *Si* $K' \in K^{\perp\!\!\!\perp}\alpha$ *y* $K' \neq \emptyset$ *entonces* $K' \cap \sigma(K^{\perp\!\!\!\perp}\alpha) \neq \emptyset$.

La función de incisión selecciona que sentencias serán descartadas. El resultado de contraer un conjunto K con respecto a una creencia α debería consistir de todos los elementos del conjunto original K que no son removidos por la función de incisión.

Definición 4.6 *Hansson [Han94] Sea* K *un conjunto de sentencias,* α *una sentencia cualquiera y* $K^{\perp\!\!\!\perp}\alpha$ *el conjunto de* α*-kernels de* K*. Sea "*σ*" una función de incisión para* K*. El operador "*$-_\sigma$*", denominado* kernel contraction determinado por "σ" *se define de la siguiente manera:*

$$K-_\sigma\alpha = K \setminus \sigma(K^{\perp\!\!\!\perp}\alpha)$$

En el caso límite en que α sea una sentencia tautológica entonces $\alpha \in Cn(\emptyset)$ y, por Definición 4.4, tenemos que $K^{\perp\!\!\!\perp}\alpha = \{\emptyset\}$. Luego, por Definición 4.6, obtenemos que $\sigma(K^{\perp\!\!\!\perp}\alpha) = \emptyset$ y $K-_\sigma\alpha = K \setminus \sigma(K^{\perp\!\!\!\perp}\alpha) = K$. En el otro caso límite, cuando $\alpha \notin Cn(K)$, sucede que $K^{\perp\!\!\!\perp}\alpha = \emptyset$ y nuevamente tenemos que $\sigma(K^{\perp\!\!\!\perp}\alpha) = \emptyset$ y $K-_\sigma\alpha = K \setminus \sigma(K^{\perp\!\!\!\perp}\alpha) = K$. Como podemos ver, ambos casos límite son tratados como en una operación de *partial meet contraction.*

Propiedades

A continuación daremos los postulados de racionalidad para el operador de *Kernel contraction* enunciados en [Han94, Han99]. Sea K una base de creencias. El operador de contracción será representado por "$-$".

(C'-1) *Éxito* (Success): Si $\alpha \notin Cn(\emptyset)$, entonces $\alpha \notin Cn(K-\alpha)$.

El primer postulado establece que la contracción debe ser exitosa; *esto es,* el resultado de contraer una base de creencias K por una sentencia α (que no es una tautología) debe ser una nueva base de creencias que no implica α.

(C'-2) *Inclusión:* $K-\alpha \subseteq K$.

Como $K-\alpha$ sigue de quitar algunas creencias de K sin agregar alguna creencia, es natural pensar que $K-\alpha$ no contiene creencias que no pertenecen a K.

(C'-3) *Uniformidad* (Uniformity): Si para todo $K' \subseteq K$, $\alpha \in Cn(K')$ si y sólo si $\beta \in Cn(K')$ entonces $K-\alpha = K-\beta$.

Esta propiedad establece que si dos sentencias α y β son implicadas por exactamente los mismos subconjuntos de K, luego la contracción de K por α debe ser igual a la contracción de K por β.

(C'-4) *Retención de Núcleo* (Core-retainment): Si $\beta \in K$ y $\beta \notin K-\alpha$ entonces existe un conjunto K' tal que $K' \subseteq K$, $\alpha \notin Cn(K')$ pero $\alpha \in Cn(K' \cup \{\beta\})$.

Las creencias que quitamos por contraer K por α deben ser todas tal que contribuyeron al hecho que K, pero no $K-\alpha$, implique α. Más precisamente, para β ser borrada en el proceso de formar $K-\alpha$ desde K, debe haber algún orden en el cual los elementos de K puedan ser removidos, tal que la extracción de β es el paso crucial por el cual α cesa de ser lógicamente implicada.

Teorema 4.4 *Hansson [Han94] El operador "−" para un conjunto K es un* operador de Kernel Contraction *si y solo si satisface éxito, inclusión, uniformidad y retención de núcleo.*

Este teorema da una caracterización axiomática de la operación de *Kernel Contraction*. Es muy similar al Teorema 4.3, el cual da una caracterización axiomática de las operaciones de *Partial Meet Contraction* a partir de las propiedades de éxito, inclusión, uniformidad y relevancia. Sin embargo, la caracterización del operador de *Kernel Contraction* está basado en la propiedad de retención de núcleo y no en la propiedad de relevancia. Como la propiedad de retención de núcleo es más débil que la propiedad de relevancia (si un operador de contracción satisface relevancia entonces satisface retención de núcleo, pero la recíproca no es cierta) toda operación de *Partial Meet Contraction* **es una** operación de *Kernel Contraction*.

4.3.3. Revisión

Las revisiones son operaciones de cambio que permiten agregar información a un conjunto de creencias o a una base de creencias respetando las siguientes premisas:

1. Preservar consistencia en el conjunto revisado.

2. Preservar tantas creencias del conjunto original como sea posible.

Por lo tanto, la revisión tiene la propiedad de que permite incorporar creencias a un conjunto de creencias generando un nuevo conjunto de creencias consistente (salvo en casos límite en los que la sentencia a incorporar sea contradictoria). Esta característica hace que la operación de revisión sea la más interesante dentro de la teoría de cambio de creencias, así como en las ciencias de la computación.

La revisión es una de las operaciones más comunes que se aplican al conocimiento cuando el mismo es utilizado en entornos dinámicos. Dado el conocimiento expresado mediante un conjunto K, al producirse cierta entrada epistémica α, el primero deberá ser revisado para determinar cuáles de las creencias en K están en desacuerdo con α, eliminar algunas de ellas, para luego incorporar α y producir un nuevo conjunto K'. Por ejemplo, si en nuestro conocimiento tenemos que Tweety vuela porque Tweety es un pájaro, al recibir la entrada epistémica que dice que los pájaros que son pingüinos no vuelan, y que Tweety es un pingüino, el conocimiento revisado no solo debería dejar de reflejar que Tweety vuela, sino que también debería denotar que Tweety no vuela.

A diferencia de las expansiones, las cuales son operaciones de cambio monótonas, las revisiones tienen la propiedad de ser operaciones de cambio no monótonas. Esto es, en la lógica clásica, si a partir de un conjunto K se deduce una sentencia (*esto es,* $K \vdash \alpha$) entonces se satisface la propiedad de monotonía, es decir, cualquier sentencia agregada a K no invalida anteriores derivaciones de α (*esto es,* $K \cup \{\beta\} \vdash \alpha$), incluso si entre las nuevas sentencias está $\neg\alpha$. Sin embargo, en los sistemas de razonamiento no monótono esta propiedad no se satisface. En un sistema de razonamiento no monótono, si un conjunto K deduce una sentencia α, entonces no es seguro que se satisfaga que el conjunto K junto con la nueva sentencia incorporada siga deduciendo a α. Precisamente, la operación de revisión se la considera una operación de cambio no monótona puesto que, si $K \vdash \alpha$ y "$*$" es un operador de revisión, entonces no es necesariamente cierto que $K * \beta \vdash \alpha$.

4.4. Otras operaciones

Hasta el momento, en el presente capítulo se han tratado aquellas operaciones básicas de la teoría de cambio de creencias, que han sido las piedras fundamentales en la vasta mayoría de los desarrollos en el área. Sin embargo, es importante destacar que no son

éstas las únicas operaciones estudiadas en cambio de creencias; otras operaciones se han desarrollado a partir de las mismas, tanto considerando casos particulares como generalizando las mismas para conseguir distintos efectos. A continuación presentaremos tales operaciones, a saber: *consolidación de creencias*, *combinación de creencias* e *integración de creencias*.

4.4.1. Consolidación de creencias

Una de las operaciones más importantes derivadas de las operaciones básicas es la de *consolidar* el conocimiento o las creencias de un individuo cuando el mismo es inconsistente. La inconsistencia en conocimiento es admitido como un problema de particular importancia [GCS10, HvHH$^+$05, HvHtT05, BQL07], especialmente cuando el mismo debe ser explotado por procesos automáticos de razonamiento o cuando el mismo se obtiene de diferentes fuentes [BHP09, BKM91, AK05]. Hay dos formas principales de aplicar procesos de razonamiento en entornos inconsistentes:

- Utilizar métodos de razonamiento tolerantes a inconsistencias, que son aquellos que pueden obtener conclusiones consistentes a partir de conocimiento inconsistente. Entre los ejemplos más importantes de tales métodos podemos citar a la *Argumentación* y el *Razonamiento Rebatible* para conocimiento expresado en formalismos como Lógica Proposicional y afines [Sim89, SL92, DW09, GS04, MWF10, Nut88, Pol87, Pra10], o Resolución Consistente de Consultas (Consistent Query Answering - CQA) y métodos derivados de la misma para aplicación en bases de datos relacionales y otros lenguajes expresivos utilizados en tecnologías como Web Semántica [ABC99, LMS12, LLR$^+$10].
- Resolver de alguna forma los problemas de inconsistencia en el conocimiento y luego aplicar métodos de razonamiento clásicos sobre el nuevo conocimiento libre de conflictos.

Los operadores de consolidación de creencias se enfocan en el segundo enfoque de tratamiento de inconsistencias. En lugar de remover del conocimiento una fórmula particular o revisar el mismo en base a una entrada epistémica, el objetivo de un operador de consolidación es tomar un conjunto inconsistente de sentencias y resolver cada problema de consistencia en el mismo, a fin de obtener un nuevo conocimiento libre de conflictos que

pueda ser explotado directamente por cualquier método de razonamiento, sin importar si el mismo es o no tolerante a inconsistencias.

4.4.2. Combinación de creencias

Tanto en los campos de la inteligencia artificial como en las bases de datos se ha vuelto muy común el tener que recurrir a conocimiento provisto por diversas fuentes, especialmente con el advenimiento de entornos altamente colaborativos como ser la Web Semántica [BLHL01]. Sin embargo, es frecuente que al considerar en conjunto conocimiento provisto por fuentes diferentes aparezcan problemas de consistencia.

Una operación de *combinación de creencias* es aquella que obtiene conocimiento consistente a partir de la unión de información potencialmente inconsistente provista por distintas fuentes. Básicamente, podemos descomponer a la operación de combinación en dos suboperaciones:

(a) Reunir toda la información provista por las distintas fuentes en un único cuerpo de información, generalmente realizado mediante la operación de unión de conjuntos.

(b) Resolver de alguna forma los conflictos que aparezcan en el conocimiento unificado (es decir, *consolidar* el mismo).

Los ejemplos más importantes de operadores de combinación son aquellos dados por Baral *et al.* en [BKM91]. Los mismos están basados en la unión de todos los conjuntos de creencias, y en la selección de subconjuntos maximalmente consistentes a través del uso de algún orden dado.

4.4.3. Integración de creencias

Finalmente, otra operación cuyo objetivo es el de unificar de manera consistente conocimiento provisto por diversas fuentes es aquella de *integración de creencias*. Si bien claramente el poseer un objetivo común hace que los operadores de combinación e integración tengan cierta similitud, también tienen una diferencia muy importante, como remarca Konieczny en [Kon00]. Al considerar directamente la unión del conocimiento los operadores de combinación pierden de vista a las fuentes que han provisto las distintas

piezas de información, por lo cual no puede realizarse análisis que consideren a las mismas cuando se están resolviendo conflictos. En el caso de los operadores de integración, en cambio, siempre es posible identificar la procedencia de las distintas piezas de información con las que se cuenta; por lo tanto, en la resolución de conflictos es posible considerar no sólo las características intrínsecas a la información sino también otras informaciones que pueden ser de utilidad, como por ejemplo la cantidad de fuentes que soportan unas pieza de información particular, o la credibilidad asociada a las mismas.

El principal desarrollo en operadores de integración se debe a Konieczny y Pino-Pérez, quienes en su trabajo seminal [KP02] sentaron las bases más importantes respecto de los operadores de integración, proporcionando postulados que expresan las propiedades esperadas de los mismos, y por lo tanto proveyendo un marco formal para su desarrollo y estudio. Konieczny y Pino-Pérez proporcionan además en [KP02] una construcción posible para operadores de integración basados en el uso de preórdenes totales del conocimiento provisto por las diversas fuentes. Otros operadores de integración de creencias para conocimiento expresado en diferentes formalismos han sido definidos en la literatura de la teoría de cambio de creencias, *v.g.,* [KP11, LM98, LM99, LS98, EKM08, HPW09, DSTW09, AK05].

4.5. Relaciones entre operaciones

Hasta el momento hemos visto diversos operadores que han surgido en la teoría de cambio de creencias por separado, dando las características y propiedades particulares de los mismos. Sin embargo, a pesar de ser definidos independientemente ciertos operadores guardan una muy estrecha relación. A continuación ahondaremos en las relaciones que unen a ciertas clases de operadores, identificando como en ocasiones un operador de una clase puede utilizarse como base para definir operadores alternativos de otra clase.

4.5.1. Revisión a partir de contracción y expansión

Como se ha mencionado anteriormente, una revisión es una operación que permite agregar consistentemente una creencia a un conjunto de creencias. Por ejemplo, si se desea incorporar una sentencia α a K debe eliminarse toda posible derivación de $\neg\alpha$ en K (si es que existe). Por lo tanto, una operación de revisión es más compleja que

una operación de contracción ya que involucra el agregado y la eliminación de creencias. De esta manera, podemos notar que las revisiones tienen una estrecha relación con las contracciones. Más aún, existen varios operadores de revisión conocidos que están definidos a partir de diferentes operaciones de contracción mediante la aplicación de la identidad de Levi. Esto es, para revisar un conjunto de creencias K con respecto a una sentencia, es posible descomponer el proceso en dos operaciones de cambio:

- Contraer K con respecto a $\neg\alpha$, para eliminar toda posible forma de inferencia de $\neg\alpha$.
- Expandir $K - \neg\alpha$ con respecto a α.

Esta descomposición garantiza (salvo en el caso límite que la entrada epistémica sea inconsistente) que el conjunto de creencias resultante sea consistente, es decir, esté en estado de equilibrio. Esta relación entre contracciones y revisiones está determinada en la Identidad de Levi. Formalmente, la definición de la Identidad de Levi es la siguiente:

$$K * \alpha = (K - \neg\alpha) + \alpha$$

Esta identidad nos da una definición formal de la operación de revisión a partir de las operaciones de contracción y expansión. De esta manera, basado en las dos contracciones vistas en la sección anterior, si el operador "$-$" de la identidad de Levi es un operador *partial meet contraction*, el operador "*" de la identidad es una operador *partial meet revision*. Del mismo modo, si el operador "$-$" de la identidad de Levi es un operador *kernel contraction*, el operador "*" de la identidad es un operador *kernel revision*.

4.5.2. Consolidación a partir de contracción

Una manera natural de conseguir la consolidación del conocimiento representado por una base de creencias K inconsistente (tradicionalmente denotado $K!$ en la literatura de cambio de creencias) es tomar la misma, obtener los subconjuntos minimalmente inconsistentes y resolver cada uno de estos conflictos. De esta forma, una vez finalizado el proceso la base de creencias resultante es consistente. Este enfoque es conocido como *contracción por falso* [Han91a], y puede ser definido utilizando una función de incisión de kernel contraction de la siguiente manera:

Definición 4.7 *Sea K un conjunto de sentencias, $\perp$ una falacia arbitraria y $K^{\perp\!\!\!\perp}\perp$ el conjunto de $\perp$-kernels de K. Sea "σ" una función de incisión para K. El operador "$!_\sigma$", denominado* kernel consolidation determinado por "σ" *se define de la siguiente manera:*

$$K!_\sigma = K \setminus \sigma(K^{\perp\!\!\!\perp}\perp)$$

Nótese que cada $\perp$-kernel representa una forma minimal de derivar una falacia arbitraria, es decir, es un conjunto minimalmente inconsistente. Claramente, el operador de consolidación mediante contracción por falso es un caso particular de los operadores de contracción por kernel contraction cuando la entrada epistémica queda fija en una falacia arbitraria. Nótese que, al ser los operadores de consolidación un caso particular de operadores de contracción, todas las propiedades de los últimos son también aplicables a los primeros. En particular, claramente después del proceso de consolidación debe valer que $K! \nvdash \perp$, es decir, la base de creencias obtenida debe ser consistente, cumpliendo con el cometido del operador.

4.5.3. Combinación a partir de consolidación

Como anticipamos en la introducción de los operadores de combinación, hay una muy estrecha relación entre éstos y los operadores de consolidación de creencias, que surge de la necesidad de resolver los conflictos en la unión de las distintas piezas de conocimiento obtenidas de las diferentes fuentes de información. En esta tarea un operador de consolidación de creencias es de gran utilidad, brindando lo necesario para obtener un conocimiento combinado de manera consistente.

Es decir que, dado un operador de consolidación cualquiera (y por ende también a partir de contracción, utilizando la *contracción por falso*), siempre es posible definir un operador de combinación que lo utilice como estrategia de resolución de conflictos. Para poder realizar esto basta con definir el operador de combinación $\oplus$ a partir de un operador de consolidación $!$ de la siguiente manera:

Definición 4.8 *Sea $\Psi = \{K_1, K_2, \ldots, K_n\}$ un conjunto de conjuntos de sentencias, y "$!$" un operador de consolidación. El operador de combinación "$\oplus_!$" obtenido a partir de $!$ se define de la siguiente manera:*

$$\oplus_!(\Psi) = (\bigcup_{K_i \in \Psi} K_i)!$$

Evidentemente, también es posible el sentido contrario: si utilizamos un operador de combinación sobre un único conjunto de sentencias K, entonces la unión de este es sí mismo; y cómo el operador de combinación se asegura que el resultado final sea consistente entonces el resultado final sería un conjunto K' consistente obtenido a partir de K, es decir $K!$.

Nótese que, si bien en la presente tesis nos enfocamos en operadores de consolidación, debido a esta estrecha relación entre los operadores todos los resultados mostrados en el presente trabajo pueden ser adaptados para un entorno de combinación, simplemente utilizando la transformación mostrada por la Definición 4.8.

4.6. Trabajos relacionados: Operaciones de cambio en formalismos mas cercanos a Datalog$^{\pm}$

A continuación profundizaremos el estudio de otros enfoques que pueden ser utilizados en la formalización de procesos de cambio. Para ello nos enfocaremos en operadores de cambio que trabajan en formalismos de representación de conocimiento que por características (como ser su expresividad o adaptación a entornos colaborativos) se asemejan más al formalismo sobre el cual desarrollaremos los operadores en la presente tesis, Datalog$^{\pm}$.

4.6.1. Description Logics

Dentro de los formalismos más destacados entre aquellos que por sus características suelen ser utilizados en entornos colaborativos como la Web Semántica [BLHL01] se encuentran las Lógicas Descriptivas (Description Logics - DLs) [BCM$^+$03]. Debido a su creciente popularidad en los últimos años, muchos desarrollos para entornos colaborativos suelen tener al menos parte del conocimiento descripto por este formalismo: y a medida que esto ocurría se hizo cada vez más evidente la necesidad de tener métodos para resolver inconsistencias tanto para las ontologías en un punto determinado del tiempo como para la dinamicidad de las mismas mediante *evolución de ontologías* (ontology evolution)[Sto04, HS05]. A continuación introduciremos algunos de los desarrollos

más importantes de operadores de cambio para DLs, lo que mas adelante nos servirá para establecer paralelos entre éstos y el enfoque de consolidación de ontologías Datalog$^{\pm}$ propuesto en la presente tesis.

Uno de los aportes más importantes dentro de los trabajos que usan teoría de cambios para resolver conflictos en DLs es el Qi *et al.* en [QLB06], el cual es basado en el modelo AGM [AGM85, Gär88]. El aporte de este trabajo es doble: por un lado, en él se introducen adaptaciones de los postulados AGM para su uso en entornos de Lógicas Descriptivas, proveyendo el marco formal de estudio de desarrollos para tal formalismo; por el otro, los autores definen dos operadores en bases de creencias y muestran que los mismos satisfacen los postulados propuestos. Los operadores propuestos por Qi *et al.* están basados en una técnica conocida como *debilitamiento de fórmulas* (formulæ weakening). La idea detrás del debilitamiento de fórmulas es que, por ejemplo, cuando una regla de conocimiento (conocidas en DLs como General Concept Inclusion - GCI) produce algún conflicto en lugar de remover totalmente la regla se procede a añadir excepciones a las mismas, utilizando para ello los individuos que producen los conflictos. De esta forma, no es necesario perder todo el conocimiento asociado a la misma.

Otro trabajo digno de mención en la recapitulación de desarrollos de operadores de cambio sobre DLs es [MLB05]. En el mismo dos técnicas de integración de conocimiento comúnmente utilizadas para Lógica Proposicional son adaptadas y refinadas para la expresividad de DLs. El método de resolución de conflictos está basado en tomar las bases de creencias originales y producir a partir de las mismas una Base de Conocimiento Disjuntiva (Disjunctive Knowledge Base - DKB) como resultado de la integración. Una desventaja conocida de las DKBs es que las mismas describen las posibles opciones que podemos tomar cuando se espera que conocimiento conflictivo sea explotado por un proceso de razonamiento, en lugar de elegir una opción particular.

4.6.2. Logic Programming

Otro formalismo muy extendido de representación de conocimiento es el de la Programación Lógica (Logic Programming) [Llo87, NM95, Gel08]. Al igual que en el caso de DLs, muchos operadores de cambios sobre conocimiento expresado en programas lógicos han sido desarrollados en los últimos tiempos para paliar problemas aparejados a inconsistencias. Muchos de los mismos están enfocados en la integración de programas lógicos,

donde la resolución de inconsistencias juega en rol crucial en la obtención del programa lógico integrado final.

Un importante exponente de tales trabajos es [HPW09], donde los autores introducen un proceso de integración basado en la semántica de modelos estables, mediante la utilización de la lógica de Here-and-There [Tur03]. La estrategia de integración propuesta por Hué *et al.* considera el uso de preórdenes entre los candidatos a remover para resolver los conflictos (llamados conjuntos potencialmente removidos - potential removed sets), para, de esta forma, determinar cuales de los candidatos son efectivamente removidos en la resolución de inconsistencias. Sin embargo, cabe destacar que Hué *et al.* no proveen una forma de obtener los preórdenes a utilizar: en su lugar, los autores asumen que para cualquier estrategia P hay un preorden dado que la define.

Otro trabajo notorio en el campo de la Programación Lógica es el de [DSTW09]. En el mismo se proponen dos enfoques distintos para la integración de programas lógicos. El primero sigue un enfoque de arbitración (arbitration), seleccionando aquellos modelos de un programa que varían minimamente con respecto a los modelos del resto de los programas. En tal enfoque los autores consideran insatisfacibilidad de programas (en un todo) en lugar de algún concepto en la unión de programas, y la estrategia para resolver tal insatisfacibilidad de un programa es simplemente dejar el programa totalmente fuera de consideración para la integración. El segundo enfoque, en cambio, se basa en la selección de los modelos de un programa especial P_0 que varían mínimamente con respecto a los programas a integrar, donde P_0 puede ser visto como las restricciones que guían el proceso de integración. Este enfoque puede ser visto como un caso particular del propuesto por Konieczny y Pino-Pérez en [KP02].

4.7. Resumen

El principal objetivo de la teoría de cambio es tratar de modelar la dinámica del conocimiento, esto es, como éste cambia para adaptarse a nueva información. En el presente capítulo hemos presentado los avances más importantes en la teoría, enfocándonos en las distintas clases de operaciones de cambio que han sido definidas a través de los años.

Tradicionalmente, los desarrollos en el área se enfocaron en tres operadores de cambio básicos: expansión, contracción y revisión. Expansión es la forma más simple de cambio,

y solamente involucra agregar la nueva pieza de información al conocimiento previo, sin importar la consistencia del conjunto final obtenido. Contracción se refiere al hecho de "olvidar" conocimiento, *esto es,* eliminar sentencias del conocimiento. Finalmente, la revisión se refiere al proceso de agregar nuevo conocimiento de forma que la consistencia del estado epistémico final sea asegurada, donde podemos tener operadores priorizados o no, dependiendo de si la entrada epistémica debe ser necesariamente aceptada o si puede no pertenecer al conocimiento final.

Uno de los características que más sobresalen en el área es que las diferentes clases de operadores pueden ser presentados en dos maneras: dando una construcción explícita (*v.g.,* a través de algoritmos) para el operador, o dando un conjunto de postulados de racionalidad a ser satisfechos, que determinan restricciones que los operadores deben satisfacer. Es decir, los postulados tratan a los operadores como cajas negras, describiendo sus comportamientos con respecto a la entrada en casos básicos, pero no los mecanismos internos usados, que son descriptos por las construcciones. Ambas partes son generalmente conjugadas a través de teoremas de representación que establecen relaciones biunívocas entre una construcción y el conjunto de propiedades que la misma satisface, de forma tal que que los mismos quedan interdefinidos entre sí.

En el presente capítulo hemos visto además como nuevos operadores han surgido en la teoría de cambio de creencias con el correr de los años. En particular, una de las operaciones más importantes derivadas de las operaciones básicas es la de *consolidar* el conocimiento o las creencias de un individuo cuando el mismo es inconsistente. Intuitivamente, los operadores de consolidación de creencias se enfocan en tratar de recuperar la consistencia en un conjunto de creencias que (posiblemente) ya es inconsistente. Es decir, en lugar de remover del conocimiento una fórmula particular o revisar el mismo en base a una entrada epistémica, el objetivo de un operador de consolidación es tomar un conjunto inconsistente de sentencias y resolver cada problema de consistencia en el mismo, a fin de obtener un nuevo conocimiento libre de conflictos. De esta forma, el conocimiento consolidado a través de tales operadores puede ser explotado directamente por cualquier método de razonamiento, algo que no es siempre posible en el caso de conocimiento inconsistente.

A su vez, también hemos visto como otros desarrollos en el área se enfocan en definir maneras de considerar al mismo tiempo información proveniente de diversas fuentes de información. Tal tarea no es trivial, ya que el surgimiento de inconsistencias es incluso

más frecuente en entornos colaborativos que en aquellos donde consideramos sólo una base de conocimiento. Dentro de esta clase de operadores tenemos dos corrientes.

- Una operación de *combinación de creencias* es aquella que obtiene conocimiento consistente a partir de la unión de información potencialmente inconsistente provista por distintas fuentes, *esto es,* estos operadores consideran primeramente la unión de las bases de conocimiento en juego, y luego proceden a resolver todos aquellos conflictos que hayan surgido.
- Otra posibilidad es la utilización de operadores de integración, donde siempre es posible identificar la procedencia de las distintas piezas de información con las que se cuenta; por lo tanto, en la resolución de conflictos es posible considerar no sólo las características intrínsecas a la información sino también otras informaciones que pueden ser de utilidad, como por ejemplo la cantidad de fuentes que soportan unas pieza de información particular, o la credibilidad asociada a las mismas.

Finalmente, hemos analizado cómo distintos operadores pueden ser utilizados como base para el desarrollo de otros operadores. En particular, una relación que es particularmente interesante a los fines de la presente tesis es la posibilidad de definir procesos de consolidación a través del uso de operadores de contracción. Esto es, una manera natural de conseguir la consolidación del conocimiento representado por una base de creencias K inconsistente es tomar la misma y resolver cada uno de los conflictos de inconsistencia. De esta forma, una vez finalizado el proceso la base de creencias resultante es consistente. Este enfoque es conocido como *contracción por falso*, y se basa en que cada $\bot$-kernel representa una forma minimal de derivar una falacia arbitraria, es decir, es un conjunto minimalmente inconsistente. Por lo tanto, al contraer esta falacia arbitraria claramente después del proceso de consolidación debe valer que $K! \nvdash \bot$, es decir, la base de creencias obtenida debe ser consistente, ya que al ser los operadores de consolidación un caso particular de operadores de contracción todas las propiedades de los últimos son también aplicables a los primeros. Como veremos en los Capítulos 5 y 6, los operadores de consolidación de ontologías Datalog$^{\pm}$ que presentaremos en esta disertación explotan esta relación entre operadores de contracción y consolidación, estableciendo a los operadores de consolidación de ontologías como aquellos que contraen cada conflicto (en el caso de Datalog$^{\pm}$ tanto inconsistencias como incoherencias) en la ontología original.

Otra relación entre operadores que hemos presentado y que será explotada más adelante en la tesis es la posibilidad de utilizar consolidación como la base para la definición de procesos de combinación. Tal relación surge de la necesidad por parte de los operadores de combinación de resolver los conflictos en la unión de las distintas piezas de conocimiento obtenidas de las diferentes fuentes de información. En esta tarea un operador de consolidación de creencias es de gran utilidad, brindando lo necesario para obtener un conocimiento combinado de manera consistente. En el Capítulo 7 analizaremos cómo utilizar esta relación para definir operadores de combinación de ontologías Datalog$^{\pm}$ a partir de aquellos operadores presentados en los Capítulos 5 y 6, además de cómo esto puede llevar a la definición de procesos de combinación para otros formalismos de representación de conocimiento como ser bases de datos relacionales.

Capítulo 5

Consolidación de ontologías Datalog$^{\pm}$

Con el paso de los años se ha vuelto muy frecuente encontrarse con ejemplos de colaboración e interacción entre diversos sistemas, especialmente desde el arribo de entornos altamente colaborativos impulsados por las potencialidades de la Web en general, y la Web Semántica [BLHL01] en particular. Un claro ejemplo del desarrollo y expansión de estos entornos colaborativos puede verse en el exponencial crecimiento del comercio electrónico (e-commerce). Como fue explicado anteriormente, en los últimos tiempos en estos entornos se ha vuelto muy popular el uso de ontologías como medio para la representación del conocimiento del dominio de aplicación (tanto intensional como extensional) utilizado por los sistemas colaboradores, y dentro de los distintos formalismos disponibles Datalog$^{\pm}$ es atrayente por sus propiedades de decidibilidad y tratabilidad en entornos masivos de datos.

Sin embargo, la mencionada interacción trae aparejado un problema importante: tanto el mantenimiento colaborativo del conocimiento como la combinación de datos provistos por diferentes sistemas colaboradores en conjunto suele afectar la consistencia (o la coherencia) de tal conocimiento. Como fue explicado en el Capítulo 3 de la presente tesis, en el caso particular de Datalog$^{\pm}$ tales inconsistencias representan violaciones a las NCs y EGDs impuestas a los datos.

La resolución de tales problemas de inconsistencia e incoherencia se hace una necesidad cada vez más importante a medida que las Ciencias de la Computación y los usos derivados de las misma evolucionan. Es importante definir métodos para el manejo de los

problemas de inconsistencia e incoherencia de ontologías. En el presente capítulo de la tesis introducimos un enfoque novel para la consecución de la consolidación de ontologías Datalog$^{\pm}$. El enfoque está basado en el uso de un operador de Contracción por medio de Kernel Contraction [Han94] que consolida ontologías realizando *contracción por falso* (contraction by falsum) [Han91a, Was99] en la resolución de inconsistencias, y un proceso análogo en la resolución de incoherencias.

5.1. Definición del entorno de consolidación

En [FH99] Friedman y Halpern dan cuenta de cómo ningún proceso de revisión (en este caso, consolidación) puede ser completamente descripto sin detallar previamente cierta información acerca del mismo. Por ejemplo, el modelado del estado epistémico es una variable importante a considerar para tales procesos. Esto puede verse reflejado de diversas maneras. Por ejemplo, ciertas propiedades de un proceso pueden ser adecuadas para su utilización en Lógica Proposicional, pero no así en Lógicas de Primer Orden (o fragmentos de la misma). Algo similar ocurre cuando el entorno de trabajo de procesos de revisión o consolidación sobre conjuntos de creencias se cambia a bases de creencias.

Como hemos dicho previamente, para la presentación de los operadores básicos de la teoría de revisión de creencias por razones de simplicidad hemos utilizado un entorno de Lógica Proposicional. Sin embargo, los operadores de consolidación que presentaremos en esta tesis trabajan sobre ontologías Datalog$^{\pm}$. En el Capítulo 3 hemos presentado las generalidades del lenguaje, en particular cómo el conocimiento es representado en el mismo, comenzando con la descripción del entorno de trabajo de los operadores. En la presente sección completaremos la descripción de este entorno definiendo el modelo epistémico a utilizar. Una vez completa la descripción del entorno proceredemos a introducir las propiedades que se espera satisfagan los operadores de consolidación de ontologías Datalog$^{\pm}$, para contar con un marco formal de análisis de los operadores que se presentaran luego.

5.1.1. Modelo epistémico

En el Capítulo 3 hemos adelantado parte de la descripción del entorno en el que nuestros operadores de consolidación son aplicados al presentar Datalog$^{\pm}$, el lenguaje de desarrollo de las ontologías sobre las cuales el operador resolverá conflictos.

En el presente trabajo el conocimiento a consolidar es representado a traves de una ontología Datalog$^{\pm}$. Es decir, los operadores de consolidación que presentaremos trabajan tomando un estado epistémico representado como una ontología Datalog$^{\pm}$, posiblemente incoherente e inconsistente. Es importante recordar que, al contrario de formalismos como las bases de datos relacionales, Datalog$^{\pm}$ utiliza una suposición de mundo abierto mediante la utilización de valores nulos para representar información desconocida. Esto implica que relaciones de derivación de nueva información como el Chase sean susceptibles de ser infinitas. Por lo tanto, hay que tener cuidado cuando se utilizan relaciones de consecuencia en este entorno. Como fue explicado en la Sección 4.2, dependiendo del tipo de base de conocimiento utilizada en los procesos de revisión y consolidación hay dos líneas de trabajo definidas, conocidas como modelo de coherencia (cuyo mayor ejemplo es [AGM85]) y modelo fundacional (que encontramos por ejemplo en [KM91, KM92, Fuh91, Han94, Han97b, Han01]). En el presente trabajo hemos escogido seguir el modelo fundacional para la consolidación de ontologías Datalog$^{\pm}$. Este es el enfoque utilizado en la presente tesis, donde las ontologías Datalog$^{\pm}$ no necesariamente están cerradas bajo una relación de conclusión. Con respecto a las entradas epistémicas, al utilizar operadores de consolidación no existe une entrada epistémica particular (como sí lo hay por ejemplo en revisión, donde tenemos una fórmula particular que guía el proceso).

Dada una ontología KB en el resto de la disertación asumimos una relación (arbitraria) $\preccurlyeq$ entre las fórmulas de KB. Tal relación se asume total, reflexiva, antisimétrica y transitiva. Para ganar generalidad, en el presente trabajo no restringimos tal relación a una particular; sin embargo, asumimos que para *KBs* sintácticamente iguales las relaciones son iguales. Esperamos que tal orden provenga de información natural en la ontología, pero cualquier elección arbitraria sería suficiente; claramente, $\preccurlyeq$ puede ser dada explícitamente por el usuario o ser definida para modelar algún aspecto útil en el contexto en el cual la ontología es explotada. Por ejemplo, aspectos cualitativos como el grado de confianza para fórmulas particulares, o aspectos cuantitativos como la cantidad de átomos perdidos (es decir, que ya no pueden ser inferidos) cuando removemos cierta fórmula.

5.1.2. Propiedades esperadas en un operador de consolidación - postulados

Como hemos dicho en el Capítulo 4, desde la aparición del trabajo de Alchourrón *et al.* [AGM85], es muy común en la teoría de cambio de creencias caracterizar tales procesos mediante la definición de *Postulados* que indican *qué* restricciones debe satisfacer un operador, sin especificar el *cómo* hacerlo. En la presente sección introducimos el conjunto de propiedades esperadas para un operador de consolidación de ontologías Datalog$^{\pm}$. La mayoría de los postulados presentados son adaptaciones de algunas de las propiedades propuestas por Hansson en [Han94] y por Konieczny y Pino-Pérez en [KP02]; exceptuando Congruencia, el cual fue introducido por Fuhrmann en [Fuh97]. Los postulados de Optimidad Local para Restricciones y Optimidad Local para Datos son aportes de la presente tesis que utilizaremos para enfocarnos en aquellas consolidaciones que minimizan la pérdida de información con respecto a $\preccurlyeq$. La minimización de pérdida de información es considerada localmente, *esto es,* para cada conjunto minimalmente inconsistente/incoherente.

Por el resto de la tesis denotaremos con $KB = (D, \Sigma)$ a la ontología Datalog$^{\pm}$ original que está siendo consolidada. Además, $\Upsilon(KB)$ denota la ontología Datalog$^{\pm}$ $\Upsilon(KB) = (D', \Sigma')$ resultante de la consolidación de KB, donde D' y Σ' son los componentes D y Σ en $\Upsilon(KB)$, respectivamente consolidados. En caso de necesidad (*v.g.,* cuando se esté comparando la consolidación de distintas *KBs*) diferenciaremos a las distintas *KBs* mediante el uso de subíndices. En tales casos, dada KB_i denotaremos su consolidación mediante $\Upsilon(KB_i) = (D'_i, \Sigma'_i)$.

A continuación introducimos los Postulados que se espera sean satisfechos por operadores de consolidación de ontologías Datalog$^{\pm}$. Sea Θ el conjunto de todas las ontologías Datalog$^{\pm}$; un operador de consolidación de ontologías Datalog$^{\pm}$ $\Upsilon : \Theta \longrightarrow \Theta$ debe satisfacer las siguientes propiedades:

(**OCP** 1.) **(Inclusión)** $\Sigma' \subseteq \Sigma$ y $D' \subseteq D$.

El proceso de consolidación sólo incluye en la ontología resultante fórmulas que pertenecen a la ontología original.

(**OCP** 2.) **(Éxito para Restricciones)** Para todo $X \subset \Sigma_{_T}$ tal que X es insatisfacible con respecto a $\Sigma_{_E} \cup \Sigma_{_{NC}} \subset \Sigma$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{_E} \cup \Sigma_{_{NC}} \subset \Sigma$ existe $\alpha \in X$ tal que $\alpha \notin \Sigma'$.

Todo conflicto de coherencia es atendido por el operador.

(**OCP** 3.) **(Éxito para Datos)** Para todo $X \subset D$ tal que $\text{mods}(X, \Sigma') = \emptyset$ y todo $X' \subsetneq X$ es tal que $\text{mods}(X', \Sigma') \neq \emptyset$ existe algún $\alpha \in X$ tal que $\alpha \notin D'$.

Todo conflicto de consistencia es atendido por el operador.

(**OCP** 4.) **Vacuidad para Restricciones** Si no existe $X \subset \Sigma_{_T}$ tal que X es satisfacible con respecto a $\Sigma_{_E} \cup \Sigma_{_{NC}} \subset \Sigma$ y $X \cup \alpha$ es insatisfacible con respecto a $\Sigma_{_E} \cup \Sigma_{_{NC}} \subset \Sigma$, entonces $\alpha \in \Sigma'$.

Toda TGD que no está directamente involucrada en un conflicto de coherencia es retenida.

(**OCP** 5.) **Vacuidad para Datos** Si no existe $X \subset D$ tal que $\text{mods}(X, \Sigma') \neq \emptyset$ y $\text{mods}(X \cup \alpha, \Sigma') = \emptyset$, entonces $\alpha \in D'$.

Todo átomo que no está directamente involucrado en un conflicto de consistencia es retenido.

(**OCP** 6.) **Optimalidad Local para Restricciones** Si $\alpha \in \Sigma_{_T}$ y $\alpha \notin \Sigma'$, entonces existe $\mu \subseteq \Sigma_{_T}$ tal que μ es satisfacible y $\alpha \cup \mu$ es insatisfacible con respecto a $\Sigma_{_E} \cup \Sigma_{_{NC}} \subset \Sigma$ y para todo $\beta \in \mu$ se verifica que $\alpha \lessdot \beta$.

Toda TGD que no sea retenida en el proceso de consolidación es la menos preferida en algún conflicto.

(**OCP** 7.) **(Optimalidad Local para Datos)** Si $\alpha \in D$ y $\alpha \notin D'$, entonces existe $\mu \subseteq D$ tal que $\text{mods}(\mu, \Sigma') \neq \emptyset$ y $\text{mods}(\alpha \cup \mu, \Sigma') = \emptyset$ y para todo $\beta \in \mu$ se verifica que $\alpha \lessdot \beta$.

Cada átomo que no sea retenido en el proceso de consolidación es el menos preferido en algún conflicto.

Nótese que los postulados que versan en coherencia (*esto es,* **Éxito para Restricciones, Vacuidad para Restricciones** y **Optimalidad Local para Restricciones**) usan los datos y las restricciones en la ontología original que está siendo consolidada; mientras que aquellas que manejan inconsistencias (**Éxito para Datos, Vacuidad para Datos** y **Optimalidad Local para Datos**) usan el conjunto consolidado de restricciones. Esto es, los postulados están inspirados en postulados clásicos dentro de la literatura de Revisión de Creencias, pero adaptados a su uso en entornos con características ontológicas:

no sólo los mismos tienen dos versiones (una por cada tipo de conflicto) sino que también reconocen el hecho de que incoherencia puede ser resuelta de manera independiente (y previamente) a la inconsistencia.

En base al conjunto de propiedades básicas recién introducido se pueden obtener propiedades adicionales que se siguen del mismo.

(**OCP** 8.) **(Consistencia)** $\Upsilon(KB)$ es consistente.

La ontología obtenida de la consolidación debe ser consistente, *esto es,* no debe existir una NC o una EGD que sea violada cuando aplicamos todas las TGDs en Σ' a los átomos en D' y, por lo tanto, $\text{mods}(\{D', \Sigma'\}) \neq \emptyset$.

(**OCP** 9.) **(Coherencia)** $\Upsilon(KB)$ es coherente.

La ontología obtenida de la consolidación debe ser coherente, *esto es,* el componente $\Sigma_{_T}$ en Σ' debe ser satisfacible con respecto a $\Sigma_{_{NC}} \cup \Sigma_{_E}$ en Σ'.

(**OCP** 10.) **(Congruencia)** Si $KB_1 = KB_2$, entonces $\Upsilon(KB_1) = \Upsilon(KB_2)$.

Si dos ontologías son iguales, entonces su consolidación debe coincidir.

La relación entre algunos de los postulados se presenta en la siguiente proposición.

Proposición 5.1 *Las implicaciones entre postulados es como sigue:*

- *Si* Υ *es un operador de consolidación de ontologías Datalog*$^{\pm}$ *que satisface* ***Inclusión*** *y* ***Éxito para Datos****, entonces* Υ *satisface* ***Consistencia****.*
- *Si* Υ *es un operador de consolidación de ontologías Datalog*$^{\pm}$ *que satisface* ***Inclusión*** *y* ***Éxito para Restricciones****, entonces* Υ *satisface* ***Coherencia****.*
- *Si* Υ *es un operador de consolidación de ontologías Datalog*$^{\pm}$ *que satisface* ***Éxito para Restricciones****,* ***Vacuidad para Restricciones****,* ***Optimalidad Local para Restricciones****,* ***Éxito para Datos****,* ***Vacuidad para Datos*** *y* ***Optimalidad Local para Datos*** *entonces* Υ *satisface* ***Congruencia****.*

Demostración: ver Apéndice A [página 194].

Los postulados propuestos tratan de capturar la noción de que los cambios hechos en la ontología resultante respecto de la original son necesarios, y que la ontología resultante es tanto coherente como consistente. Esto es, dada la ontología original, el proceso de consolidación sólo eliminará restricciones (TGDs) y átomos si los mismos están de alguna forma involucrados en hacer a la ontología original incoherente/inconsistente.

Nótese que los postulados de Optimalidad consideran la cantidad de pérdida de información para decidir qué elementos son eliminados de una ontología. Por lo tanto, un operador de consolidación es en tal sentido dependiente de la forma sintáctica de la KB, lo cual, como se comentó en el Capítulo 4, es esperado para enfoques que utilizan bases de creencias, ya que los mismos son intrínsicamente dependientes de la sintaxis y pueden producir diferentes resultados para ontologías semánticamente equivalentes. Por ejemplo, en el presente trabajo asumimos que todas las TGDs tienen sólo un átomo en su cabeza. Esta suposición no genera pérdida de generalidad; sin embargo, la diferencia en la forma sintáctica de una TGD con más de un átomo en su cabeza respecto de su transformación en TGDs con sólo un átomo en la cabeza afectará el resultado final de la consolidación.

5.2. Operador de consolidación de ontologías Datalog$^{\pm}$ basado en Kernel Contraction

En el Capítulo 3 hemos presentado ejemplos de como inconsistencias e incoherencias pueden aparecer en ontologías Datalog$^{\pm}$. Además, establecimos un conjunto de propiedades que se espera sean cumplidas por un operador de consolidación para que el mismo realice cambios *adecuados* a la ontología a consolidar y así obtener una ontología consolidada $\Upsilon(KB)$ consistente y coherente. En la presente sección introduciremos una posible construcción para un operador de consolidación con las características descriptas.

Como se ha visto en el Capítulo 4, en la literatura de la teoría del cambio de creencias muchos operadores de revisión y contracción han sido desarrollados. Una característica saliente de tales operadores es que los mismos cuentan con una entrada epistémica determinada, una fórmula (o conjuntos de fórmulas, *v.g.,* [Del11, DJ12]) que guía el proceso de cambio. Esto es algo que no ocurre en un proceso de consolidación de ontologías Datalog$^{\pm}$, ya que los mismos no revisan la base de creencias por una fórmula determinada o remueven una fórmula particular de las mismas. Sin embargo, como fue explicado en la Sección 4.5.2

hay una clara conexión entre operadores de contracción y aquellos de consolidación, al punto que la tarea de restaurar consistencia en el conocimiento es conocida también como contracción por *falso* [Han91a, Was99]. En la presente tesis explotamos tal relación definiendo el proceso de consolidación como la aplicación de funciones de incisión sobre conjuntos mínimos conflictivos en los distintos componentes de la ontología. Más específicamente, para resolver incoherencias primero encontraremos los kernels de dependencias; y luego los kernels de datos para resolver inconsistencias en D con respecto a Σ.

5.2.1. Kernels de dependencias y kernels de datos

Para definir el proceso de consolidación de manera apropiada introducimos ahora los subconjuntos de la base de creencias que son relevantes para el proceso. El proceso se enfoca en la resolución de conflictos en conjuntos conflictivos mínimos (en el sentido de inclusión de conjuntos) conocidos como *kernels*.

Primeramente, nos enfocamos en los conjuntos que traen aparejados problemas de coherencia en una ontología, denominados *kernels de dependencias*, los cuales son subconjuntos minimales del conjunto de TGDs que tienen conflicto con el conjunto de NCs y EGDs en la ontología. Intuitivamente, un kernel de dependencias es un conjunto mínimo de TGDs tal que la aplicación de las TGD en él sobre cualquier conjunto relevante de átomos provoca una violación de las restricciones en $\Sigma_{NC} \cup \Sigma_E$. Los kernels de dependencias son definidos formalmente a continuación.

Definición 5.1 (Kernels de Dependencias) *Sea $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ con $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$, donde Σ_T es el conjunto de todas las TGD, Σ_E es el conjunto de todas las EGD y Σ_{NC} es el conjunto de todas las NC en Σ. El conjunto de los kernels de dependencias de KB, notado por $\perp\!\!\!\perp_{(\Sigma,KB)}$, es el conjunto de todos los $K \subseteq \Sigma_T$ tal que K es un conjunto insatisfacible de TGDs respecto de $\Sigma_{NC} \cup \Sigma_E$ y todo subconjunto estricto K' de K ($K' \subsetneq K$) es satisfacible respecto de $\Sigma_{NC} \cup \Sigma_E$.*

Ejemplo 5.1 (Kernels de dependencias) *Considere el siguiente conjunto de restricciones en una ontología Datalog$^{\pm}$ KB:*

$$KB = \left\{ \begin{array}{ll} \Sigma_{_{NC}}: & \{\tau_1 : consejero(X) \wedge regente(X) \rightarrow \bot, \\ & \tau_2 : sin\_derecho\_gobernar(X) \wedge heredero(X)\} \\ \\ \Sigma_{_{E}}: & \{\nu_1 : aconseja(X,R) \wedge aconseja(X,R') \rightarrow R = R'\} \\ \\ \Sigma_{_{T}}: & \{\sigma_1 : aconseja(X,R) \rightarrow consejero(X), \\ & \sigma_2 : propone\_ley(X,R) \rightarrow regente(X), \\ & \sigma_3 : principe(P) \rightarrow heredero(P), \\ & \sigma_4 : hijo(P,R) \wedge rey(R) \rightarrow principe(P), \\ & \sigma_5 : consejero(C) \rightarrow regente(C), \\ & \sigma_6 : hijo\_bastardo(X,Y) \rightarrow hijo(X,Y), \\ & \sigma_7 : hijo\_bastardo(X,R) \wedge rey(R) \rightarrow sin\_derecho\_gobernar(X)\} \end{array} \right\}$$

Para esta KB existen dos kernels de dependencias, esto es,

$$\perp\!\!\!\perp_{(\Sigma,KB)} = \{\{\sigma_3, \sigma_4, \sigma_6, \sigma_7\}, \{\sigma_5\}\}.$$

Es sencillo mostrar que los kernels de dependencias para una ontología Datalog$^{\pm}$ son independientes del componente D particular en la ontología y, por lo tanto, pueden ser obtenidos considerando solamente el componente Σ, descartando la base de datos D particular. Si reemplazamos el componente D en una ontología con un conjunto vacío de átomos, entonces los kernels de dependencias para esa ontología con el conjunto vacío de átomos son los mismos que en la ontología original.

Lema 5.2 *Sean* $KB_1 = (D_1, \Sigma_1)$ *donde* $\Sigma_1 = \Sigma_{1_T} \cup \Sigma_{1_E} \cup \Sigma_{1_{NC}}$, *y* $KB_2 = (\emptyset, \Sigma_2)$ *donde* $\Sigma_2 = \Sigma_{2_T} \cup \Sigma_{2_E} \cup \Sigma_{2_{NC}}$ *dos ontologías Datalog*$^{\pm}$ *tales que* $\Sigma_1 = \Sigma_2$. *Entonces,* $\perp\!\!\!\perp_{(\Sigma_1,KB_1)} = \perp\!\!\!\perp_{(\Sigma_2,KB_2)}$.

Demostración: ver Apéndice A [página 197].

Además de la remoción de TGDs que contribuyen a hacer el conjunto Σ en una ontología insatisfacible (haciendo a la ontología en cuestión incoherente), el proceso de consolidación de ontologías Datalog$^{\pm}$ debe también remover átomos del componente D en la ontología a consolidar para poder resolver inconsistencias, si las hubiera. Por lo tanto, análogamente a la manera en que definimos kernels de dependencias, procedemos a definir *kernels de datos* como los conjuntos mínimos de átomos que hacen a una KB inconsistente.

Definición 5.2 (Kernels de Datos) *Sea* $KB = (D, \Sigma)$ *una ontología Datalog*$^{\pm}$ *con* $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$. *Entonces,* $\perp\!\!\!\perp_{(D,KB)}$ *para* D *respecto de* Σ *es el conjunto de todos los* $X \subseteq D$ *tales que* $\text{mods}(X, \Sigma) = \emptyset$ *y para todo* $X' \subsetneq X$ *se verifica que* $\text{mods}(X', \Sigma) \neq \emptyset$.

Es importante remarcar que, mientras que los kernels de dependencias son obtenidos independientemente del componente D en la ontología (ya que D no tiene influencia en la determinación de incoherencia), para obtener el conjunto de kernels de datos utilizamos un conjunto Σ fijo.

Ejemplo 5.2 (Kernels de Datos) Considere la siguiente *KB* propuesta por Lukasiewicz *et al.* en [LMS12], la cual es coherente pero inconsistente.

$$KB = \left\{ \begin{array}{ll} D: & \{dirige(john, d_1), dirige(tom, d_1), dirige(tom, d_2), \\ & supervisa(tom, john), trabaja\_en(john, d_1), trabaja\_en(tom, d_1)\} \\ \\ \Sigma_{_{NC}}: & \{supervisa(X, Y) \wedge administrador(Y) \rightarrow \bot, \\ & supervisa(X, Y) \wedge trabaja\_en(X, D) \wedge dirige(Y, D) \rightarrow \bot\} \\ \\ \Sigma_{_E}: & \{dirige(X, D) \wedge dirige(X, D') \rightarrow D = D'\} \\ \\ \Sigma_{_T}: & \{\sigma_1 : trabaja\_en(X, D) \rightarrow empleado(X), \\ & \sigma_2 : dirige(X, D) \rightarrow empleado(X), \\ & \sigma_3 : dirige(X, D) \wedge trabaja\_en(X, D) \rightarrow administrador(X)\} \end{array} \right\}$$

Para esta *KB*, el conjunto de los kernels de datos es

$$\begin{array}{c} \perp\!\!\!\perp_{(D,KB)} = \{\{supervisa(tom, john), dirige(john, d_1), trabaja\_en(john, d_1)\}, \\ \{supervisa(tom, john), dirige(john, d_1), trabaja\_en(tom, d_1)\}, \\ \{dirige(tom, d_1), dirige(tom, d_2)\}\}. \end{array}$$

En el siguiente lema mostramos un ejemplo de como, en el entorno ontológico de Datalog$^{\pm}$, la *indiscernibilidad de idénticos* de Leibniz [vL76] se verifica con respecto a los kernels en ontologías Datalog$^{\pm}$, ya que cuando dos *KBs* son sintácticamente equivalentes las mismas tienen los mismos kernels de dependencias y kernels de datos.

Lema 5.3 *Sean* $KB_1 = (D_1, \Sigma_1)$ *y* $KB_2 = (D_2, \Sigma_2)$ *dos ontologías Datalog*$^{\pm}$ *tales que* $KB_1 = KB_2$. *Entonces,* $\perp\!\!\!\perp_{(D_1,KB_1)} = \perp\!\!\!\perp_{(D_2,KB_2)}$ *y* $\perp\!\!\!\perp_{(\Sigma_1,KB_1)} = \perp\!\!\!\perp_{(\Sigma_2,KB_2)}$.

Demostración: ver Apéndice A [página 198].

Nótese que el otro sentido del Lema 5.3 (*esto es,* una versión de la *identidad de indiscernibles*, pero considerando sólo una propiedad – los kernels – como igual) no se verifica, ya que dos *KBs* pueden tener los mismos kernels, incluso cuando las mismas no son sintácticamente equivalentes, como se muestra a continuación:

Ejemplo 5.3 (Fallo de Identidad de Indiscernibles) *Considere las siguientes* KBs*:*

$$KB_1 = \left\{ \begin{array}{ll} D_1: & \{jugador(peter), da\_indicaciones(roman), en\_cancha(roman)\} \\ \Sigma_{1_{NC}}: & \{\tau_1 : jugador(X) \wedge tecnico(X) \rightarrow \bot\} \\ \Sigma_{1_T}: & \{\sigma_1 : da\_indicaciones(X) \rightarrow tecnico(X), \\ & \sigma_2 : en\_cancha(X) \rightarrow jugador(X)\} \end{array} \right\}$$

$$KB_2 = \left\{ \begin{array}{ll} D_2: & \{da\_indicaciones(roman), en\_cancha(roman), lleva\_cinta(martin)\} \\ \Sigma_{2_{NC}}: & \{\tau_1 : jugador(X) \wedge tecnico(X) \rightarrow \bot\} \\ \Sigma_{2_T}: & \{\sigma_1 : da\_indicaciones(X) \rightarrow tecnico(X), \\ & \sigma_2 : en\_cancha(X) \rightarrow jugador(X), \\ & \sigma_3 : lleva\_cinta(X) \rightarrow capitan(X)\} \end{array} \right\}$$

Claramente, $\perp\!\!\!\perp_{(\Sigma_1, KB_1)} = \perp\!\!\!\perp_{(\Sigma_2, KB_2)} = \emptyset$ *y* $\perp\!\!\!\perp_{(D_1, KB_1)} = \perp\!\!\!\perp_{(D_2, KB_2)} = \{da\_indicaciones(roman), en\_cancha(roman)\}$*, pero sin embargo* $KB_1 \neq KB_2$*.*

5.2.2. Funciones de incisión generales

Una vez que hemos identificado los kernels sobre los que aplicaremos las operaciones de contracción en pos de resolver incoherencias e inconsistencias, debemos establecer como tales problemas son resueltos. En la Sección 4.3.2, en el enfoque utilizado en Kernel Contraction, se utilizan funciones de incisión para seleccionar cuáles elementos de los kernels son removidos para resolver conflictos. Una función de incisión toma como argumento de entrada kernels de dependencias y kernels de datos y selecciona qué TGDs serán removidas de Σ_T y qué átomos serán removidos de D.

Definición 5.3 (Función de incisión general) *Sea $\mathcal{R}$ un esquema relacional y $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ sobre $\mathcal{R}$ donde $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$. Sea $\lessdot$ un orden entre las fórmulas de KB, y sea $\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de kernels de dependencias para KB, y $\perp\!\!\!\perp_{(D,KB)}$ el conjunto de kernels de datos para KB. Una* función de incisión general *para KB es una función $\delta : 2^{2^{\mathcal{L}_\mathcal{R}}} \longrightarrow 2^{\mathcal{L}_\mathcal{R}}$ tal que las siguientes condiciones se verifican:*

- $\delta(KB) \subseteq \bigcup(\perp\!\!\!\perp_{(\Sigma,KB)}) \cup \bigcup(\perp\!\!\!\perp_{(D,KB)})$.
- *Para todo $X \in \perp\!\!\!\perp_{(\Sigma,KB)}$ tal que $X \neq \emptyset$ se verifica que $(X \cap \delta(KB)) \neq \emptyset$.*
- *Para todo $X' \in \perp\!\!\!\perp_{(D,KB)}$ tal que $X' \neq \emptyset$ se verifica que $(X' \cap \delta(KB)) \neq \emptyset$.*
- *Para todo $\sigma \in \delta(KB)$ existe $X \in \perp\!\!\!\perp_{(\Sigma,KB)}$ tal que para cualquier $\sigma' \in X$ se verifica que $\sigma \lessdot \sigma'$.*
- *Para todo $\alpha \in (X' \cap \delta(KB))$ existe $X' \in \perp\!\!\!\perp_{(D,KB)}$ tal que para cualquier $\beta \in X'$ se verifica que $\alpha \lessdot \beta$.*

5.2.3. Influencia de la incoherencia en la consolidación

La Definición 5.3 dice que una función de incisión general selecciona para remoción TGDs y átomos de los kernels de dependencias y de los kernels de átomos, respectivamente, para de esta forma restaurar la consistencia y la coherencia. Cualquier función de incisión que se adecue a a la Definición 5.3 puede ser usada como base para un operador de consolidación. Sin embargo, es importante remarcar que tal operador no diferencia realmente entre restaurar coherencia y consistencia. En la literatura clásica de Revisión de Creencias esto no es un problema ya que no existe la noción de incoherencia, y no hay distinción entre las reglas y los hechos en lenguajes como Lógica Proposicional; por lo tanto, sólo conflictos de consistencia pueden aparecer, y no es necesario considerar incoherencias. Sin embargo, en entornos ontológicos como el de Datalog$^{\pm}$ tenemos la oportunidad de explotar el hecho de que tenemos dos tipos diferentes aunque relacionados de conflictos para encararlos de forma separada con el objetivo de encontrar una solución que se ajuste mejor a las necesidades de las aplicaciones que se apoyan en tales bases de conocimiento.

En la presente tesis hemos estado remarcando la importancia de diferenciar, y manejar de manera apropiada, incoherencia de inconsistencia, ya que la calidad de la ontología consolidada dependerá en gran medida de ello — siempre asumiendo que se busca la

mínima pérdida de información en el proceso. Esto es particularmente importante a la hora de definir qué constituye un kernel. Para entender mejor la importancia de esto considere el siguiente ejemplo.

Ejemplo 5.4 (Influencia de la incoherencia en la consolidación) Considere la KB del Ejemplo 3.10. Allí tenemos un $\Sigma = \Sigma^2_{_T} \cup \Sigma^2_{_E} \cup \Sigma^2_{_{NC}}$ tal que $\Sigma^2_{_T}$ es insatisfacible. Como se explicó en el Ejemplo 3.10, para el conjunto unitario $\{cantante\_rock(axl)\}$ tenemos que la NC τ_1 : $garganta\_lastimada(X) \wedge puede\_cantar(X) \rightarrow \bot$ es violada, haciendo $\{cantante\_rock(axl)\}$ inconsistente con Σ. Entonces, $\{cantante\_rock(axl)\}$ es un kernel de datos y lo mismo es verificable para cada conjunto unitario de átomos en D relevante a algún conjunto insatisfacible de TGDs. Por lo tanto, tenemos que

$$\perp\!\!\!\perp_{(D,KB)} = \left\{ \begin{array}{c} \{cantante\_rock(axl)\}, \\ \{cantante\_rock(ronnie)\}, \\ \{cantante\_rock(roy)\}, \\ \{tiene\_fans(ronnie)\} \end{array} \right\}$$

Considere el kernel $\{cantante\_rock(axl)\}$; tenemos que $\delta(\{cantante\_rock(axl)\}) = cantante\_rock(axl)$. Lo mismo ocurre con cada kernel en $\perp\!\!\!\perp_{(D,KB)}$, y por lo tanto $\delta(\perp\!\!\!\perp_{(D,KB)}) = \perp\!\!\!\perp_{(D,KB)}$.

El problema en este ejemplo es que los kernels de datos están computados respecto del componente Σ original, el cual, en este caso, contiene conjuntos insatisfacibles de restricciones. Como podemos ver en el Ejemplo 5.4, esto es de vital importancia cuando tenemos átomos relevantes a conjuntos insatisfacibles: en ese caso, cualquier función de incisión general (y cualquier técnica de manejo de inconsistencia basada en remoción que no trate conflictos de coherencia) tendrá necesariamente que remover tales átomos.

Proposición 5.4 *Sea $\mathcal{R}$ un esquema relacional, $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ sobre $\mathcal{R}$ donde $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$, y sea $\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de kernels de dependencias para KB. Sea δ una función de incisión general. Si $\alpha \in D$ es relevante a algún $X \in \perp\!\!\!\perp_{(\Sigma,KB)}$ entonces $\alpha \in \delta(KB)$.*

Demostración: ver Apéndice A [página 198].

Como corolario de la Proposición 5.4 tenemos que si cada átomo en D es relevante a algún conjunto insatisfacible entonces debemos de remover cada átomo en D para restaurar la consistencia.

Corolario 5.5 *Sea $\mathcal{R}$ un esquema relacional, $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ sobre $\mathcal{R}$ donde $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$, y sea $\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de kernels de dependencias para KB. Sea δ una función de incisión general. Si para todo $\alpha \in D$ se verifica que α es relevante a algún $X \in \perp\!\!\!\perp_{(\Sigma,KB)}$ entonces $D \subseteq \delta(KB)$.*

Demostración: ver Apéndice A [página 199].

Como puede verse, la incoherencia puede tener una gran influencia en la consolidación si no se trata apropiadamente (esto es, previa a la restauración de inconsistencia). Es mejor en tales casos computar los kernels de datos basándonos solamente en las dependencias retenidas luego de la consolidación de Σ. En el Lema 5.2 mostramos que los kernels de dependencias pueden ser obtenidos independientemente del componente D de la ontología original, ya que los conjuntos insatisfacibles son tales que los mismos violan una EGD o NC para *cualquier* conjunto relevante de átomos. Por lo tanto, podemos primero obtener $\perp\!\!\!\perp_{(\Sigma,KB)}$, y usar la función de incisión sobre los kernels de dependencias para seleccionar qué TGDs serán eliminadas. Luego, calculamos $\perp\!\!\!\perp_{(D,KB)}$ y el orden entre conjuntos de átomos $\preccurlyeq_1$ basándonos en el resultado de aplicar la función de incisión sobre $\perp\!\!\!\perp_{(\Sigma,KB)}$. De esta manera, nos enfocamos en el subconjunto del componente Σ original que prevalecerá en el proceso de consolidación.

5.2.4. Funciones de incisión en restricciones y funciones de incisión en datos

En lo subsiguiente definiremos tanto *funciones de incisión en restricciones* como *funciones de incisión en datos* las cuales son usados para seleccionar que remover de la ontología original para restaurar coherencia/consistencia, respectivamente. Comenzaremos definiendo una función de incisión que trabaje sobre los kernels de dependencias.

Definición 5.4 (Función de incisión en restricciones) *Sea $\mathcal{R}$ un esquema relacional y $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ sobre $\mathcal{R}$ donde $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$. Sea $\preccurlyeq$ un orden sobre las fórmulas en KB y $\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de kernels de dependencias para*

KB. Una función de incisión en restricciones *para KB es una función* $\rho : 2^{2^{\mathcal{L}_\mathcal{R}}} \mapsto 2^{\mathcal{L}_\mathcal{R}}$ *tal que todas las siguientes condiciones se verifican:*

- $\rho(KB) \subseteq \bigcup(\perp\!\!\!\perp_{(\Sigma,KB)})$.
- *Para todo* $X \in \perp\!\!\!\perp_{(\Sigma,KB)}$ *tal que* $X \neq \emptyset$*, se verifica que* $(X \cap \rho(KB)) \neq \emptyset$.
- *Para todo* $\sigma \in \rho(KB)$ *existe* $X \in \perp\!\!\!\perp_{(\Sigma,KB)}$ *tal que para cualquier* $\sigma' \in X$ *se verifica que* $\sigma \lessdot \sigma'$.

Intuitivamente, una función de incisión en restricciones toma kernels de dependencias y selecciona para remoción TGDs en ellos, lo que hace que una vez removidas el conjunto de TGDs resultante sea satisfacible y, por lo tanto, la ontología resultante sea coherente.

Análogamente definimos las funciones de incisión en datos para resolver inconsistencias en $\perp\!\!\!\perp_{(D,KB)}$.

Definición 5.5 (Función de incisión en datos) *Sea* $\mathcal{R}$ *un esquema relacional,* $KB = (D, \Sigma)$ *una ontología Datalog*$^{\pm}$ *sobre* $\mathcal{R}$*,* $\lessdot$ *un orden sobre las fórmulas en KB, y* $\perp\!\!\!\perp_{(D,KB)}$ *el conjunto de los kernels de datos para KB sobre* Σ*. Una* función de incisión en datos *para D es una función* $\varrho : 2^{2^{\mathcal{L}_\mathcal{R}}} \mapsto 2^{\mathcal{L}_\mathcal{R}}$ *tal que las siguientes condiciones se verifican:*

- $\varrho(KB) \subseteq \bigcup(\perp\!\!\!\perp_{(D,KB)})$.
- *Para todo* $X \in \perp\!\!\!\perp_{(D,KB)}$ *tal que* $X \neq \emptyset$ *se verifica que* $(X \cap \varrho(KB)) \neq \emptyset$.
- *Para todo* $\alpha \in (X \cap \varrho(KB))$ *existe* $X \in \perp\!\!\!\perp_{(D,KB)}$ *tal que para cualquier* $\beta \in X$ *se verifica que* $\alpha \lessdot \beta$.

Cabe remarcar que la última condición en las Definiciones 5.4 y 5.5 asegura que sea removida de cada kernel *al menos* aquella fórmula con menor importancia dentro de $\lessdot$, ya que las funciones de incisión hacen la mejor elección posible en cada kernel.

5.2.5. Operador de consolidación de ontologías Datalog$^{\pm}$

Finalmente, utilizando los conceptos introducidos procedemos a definir el operador de consolidación de ontologías Datalog$^{\pm}$ que representa las dos diferentes partes de la consolidación. Primero, la restauración de coherencia del componente Σ se hace basada en los kernels de dependencias obtenidos a partir del componente D de la ontología original. Segundo, la restauración de consistencia del componente D se hace basado en los kernels de datos obtenidos usando el conjunto de restricciones resultante de aplicar sobre Σ' una función de incisión sobre restricciones. Formalmente, el operador de consolidación de ontologías Datalog$^{\pm}$ es definido de la siguiente forma.

Definición 5.6 (Operador de consolidación basado en Kernel Contraction) *Sea KB una ontología Datalog$^{\pm}$, ρ una función de incisión en restricciones y ϱ una función de incisión en datos. Además, sea $KB^{\star} = (D, \Sigma \setminus \rho(KB))$ la ontología resultante de remover de KB aquellas TGDs seleccionadas por ρ. El operador de consolidación basado en Kernel Contraction $\Upsilon_{\rho,\varrho}(KB)$, es definido como:*

$$\Upsilon_{\rho,\varrho}(KB) = (D \setminus \varrho(KB^{\star}), \Sigma \setminus \rho(KB))$$

El resultado de $\Upsilon_{\rho,\varrho}(KB)$ es la ontología obtenida removiendo TGDs (seleccionadas por ρ de $\perp\!\!\!\perp_{(\Sigma,KB)}$) y átomos (seleccionados por ϱ de $\perp\!\!\!\perp_{(D,KB^{\star})}$) de la ontología original KB. Es importante remarcar que, por un lado, sólo TGDs son removidas de Σ, ya que los kernels de dependencias no contienen EGDs ni NCs. Por el otro lado, como la función de incisión en datos usa $KB^{\star}$ en lugar de KB entonces sólo átomos de D que estén en conflicto con $\Sigma \setminus \rho(KB)$ son removidos, ya que los kernels de datos son calculados en base a las restricciones obtenidas luego de aplicar sobre Σ la función de incisión en restricciones.

5.3. Relación entre los propiedades y la construcción: Teorema de Representación

Hasta el momento hemos introducido las propiedades que se esperan de un operador de consolidación de ontologías Datalog$^{\pm}$, presentando además una construcción posible para tal operador basada en el uso de Kernel Contraction. Estamos listos entonces para establecer la relación entre el conjunto de postulados para operadores de consolidación de

ontologías Datalog$^{\pm}$ y el operador de consolidación presentado. En lo que sigue denotaremos con $\Upsilon_{\rho,\varrho}$ un operador de consolidación definido como en la Definición 5.6 donde ρ y ϱ corresponden a funciones arbitrarias de incisión en restricciones y datos, respectivamente.

Teorema 5.1 (Teorema de Representación) *El operador $\Upsilon_{\rho,\varrho}$ es un* operador de consolidación de ontologías Datalog$^{\pm}$ basado en Kernel Contraction *para una ontología Datalog$^{\pm}$ KB si y sólo si satisface* ***Inclusión***, ***Éxito para Restricciones***, ***Éxito para Datos***, ***Vacuidad para Restricciones***, ***Vacuidad para Datos***, ***Optimalidad Local para Restricciones*** *y* ***Optimalidad Local para Datos***.

Demostración: ver Apéndice A [página 199].

Es importante remarcar que, si bien no requerimos explícitamente que un operador de consolidación de ontologías Datalog$^{\pm}$ produzca una ontología consolidada coherente y consistente, del Teorema 5.1 se sigue que un operador como el presentado en la Definición 5.6 satisface las requerimientos esperados de **Coherencia** y **Consistencia**.

Corolario 5.6 (Corolario del Teorema 5.1) *El operador $\Upsilon_{\rho,\varrho}$ satisface* ***Coherencia*** *y* ***Consistencia***.

Demostración: ver Apéndice A [página 204].

El Corolario 5.6 dice que el operador de consolidación basado en Kernel Contraction como es introducido en la Definición 5.6 computa una ontología *consolidada* coherente y consistente.

5.4. Ejemplo completo de consolidación de ontologías Datalog$^{\pm}$

En las secciones previas hemos introducido una construcción posible para un operador de consolidación de ontologías Datalog$^{\pm}$. A continuación presentaremos un ejemplo completo (inspirado en uno similar introducido en [LMS12]) donde se detalla tal proceso.

Ejemplo 5.5 (Consolidación de ontologías Datalog$^{\pm}$) Supongamos que tenemos la siguiente ontología KB (incoherente e inconsistente), la cual expresa la información que hemos podido recabar acerca del funcionamiento de cierta empresa.

$$KB = \left\{ \begin{array}{ll} D: & \{a_1 : jefe(walter), a_2 : supervisa(walter, jesse), \\ & a_3 : toma\_desiciones(walter), a_4 : toma\_desiciones(jesse), \\ & a_5 : supervisa(skyler, walter), a_6 : empleado(walter), \\ & a_7 : a\_cargo\_de(jesse, distribution), \\ & a_8 : a\_cargo\_de(walter, cooking) \\ & a_9 : en\_huelga(mike)\} \\ \\ \Sigma_{_{NC}}: & \{\tau_1 : obedece\_ordenes(X) \wedge toma\_desiciones(X) \rightarrow \bot, \\ & \tau_2 : supervisa(Y, X) \wedge supervisor(X) \rightarrow \bot, \\ & \tau_3 : ausente(X) \wedge en\_huelga(X) \rightarrow \bot\} \\ \\ \Sigma_{_{E}}: & \{\nu_1 : a\_cargo\_de(X, Y) \wedge a\_cargo\_de(X, Y') \rightarrow Y = Y'\} \\ \\ \Sigma_{_{T}}: & \{\sigma_1 : empleado(X) \rightarrow es\_supervisado(X), \\ & \sigma_2 : es\_supervisado(X) \rightarrow obedece\_ordenes(X), \\ & \sigma_3 : jefe(X) \rightarrow obtiene\_ganancias(X), \\ & \sigma_4 : supervisa(Y, X) \rightarrow supervisor(Y), \\ & \sigma_5 : supervisa(Y, X) \rightarrow empleado(X), \\ & \sigma_6 : es\_supervisado(X) \rightarrow toma\_desiciones(X), \\ & \sigma_7 : es\_supervisado(X) \rightarrow tiene\_tarea(X), \\ & \sigma_8 : tiene\_tarea(X) \rightarrow recibe\_paga(X), \\ & \sigma_9 : tiene\_tarea(X) \rightarrow \exists Y a\_cargo\_de(X, Y), \\ & \sigma_{10} : en\_huelga(X) \rightarrow ausente(X)\} \end{array} \right\}$$

Ahora, para comenzar con la primer parte del proceso de consolidación, *esto es,* con la resolución de incoherencias por medio de hacer el conjunto $\Sigma_{_T}$ satisfacible, obtenemos los kernels de dependencias para KB:

$$\perp\!\!\!\perp_{(\Sigma, KB)} = \{\{\sigma_2, \sigma_6\}, \{\sigma_{10}\}\}$$

A continuación, debemos establecer el orden entre estas TGDs. Como hemos explicado previamente en la Sección 5.1.1, en nuestra definición de operadores de consolidación no introducimos una relación $\lessdot$ particular, sino que usamos una relación general abstracta. Sin embargo, para el presente ejemplo asumiremos que en el entorno de aplicación particular en el que se están usando los operadores tal relación es usada para modelar cuanta información se pierde cuando borramos alguna fórmula en la ontología (midiendo esto como la cantidad de átomos que no pueden ser inferidos luego de la remoción), y la plausibilidad asociada a las fórmulas como desempate entre aquellas fórmulas que inducen la misma pérdida de átomos. Remarcamos, de todas formas, que esto es un simple ejemplo de cómo la relación $\lessdot$ puede ser definida; relaciones más complejas pueden ser usadas si las mismas son necesarias en entornos de aplicación particulares, ya que los operadores son definidos de forma independiente a la relación particular usada para seleccionar fórmulas para remoción. Comenzamos mostrando los átomos perdidos cuando se remueven TGDs, y luego introducimos los ordenes de preferencia entre las mismas.

- Átomos que ya no pueden ser inferidos luego de remover σ_2 : $\{obedece\_ordenes(walter), obedece\_ordenes(jesse)\}$.
- Átomos que ya no pueden ser inferidos luego de remover σ_6 : ninguno.
- Átomos que ya no pueden ser inferidos luego de remover σ_{10} : $\{ausente(mike)\}$.

Luego, el orden para estas TGDs es $\sigma_6 \lessdot \sigma_{10} \lessdot \sigma_2$. El comportamiento de la función de incisión en restricciones es el siguiente:

$$\rho(\{\sigma_2, \sigma_6\}) = \{\sigma_6\}$$
$$\rho(\{\sigma_{10}\}) = \{\sigma_{10}\}$$

Por lo tanto, ahora podemos hacer $\Sigma_{_T}$ satisfacible (y por lo tanto KB coherente) eliminando del mismo las dos TGDs seleccionadas por ρ. A continuación, comienza la segunda parte del proceso de consolidación: la resolución de inconsistencias. Como fue explicado previamente, para esta parte el operador considera solamente TGDs que efectivamente pertenecerán a la ontología final consolidada. Para el ejemplo en cuestión esto es $\Sigma'_{_T} = \Sigma_{_T} \setminus \{\sigma_6, \sigma_{10}\}$. A partir de aquí sea $KB^\star = (D, \Sigma')$; basado en $KB^\star$ calculamos los kernels de datos.

$$\perp\!\!\!\perp_{(D,KB^{\star})} = \{\{a_2, a_4\}, \{a_3, a_5\}, \{a_3, a_6\}, \{a_2, a_5\}\}$$

Siguiendo el enfoque utilizado para las TGDs, a continuación introducimos la relación $\lessdot$ particular usada en este ejemplo para calcular el orden entre átomos. De esta manera, la función de incisión en datos puede seleccionar para remoción átomos en los kernels de datos. Los átomos que no pueden ser inferidos luego de las remociones son:

- Átomos que ya no pueden ser inferidos luego de remover a_2 : $\{supervisa(walter, jesse), supervisor(walter), empleado(jesse), es\_supervisado(jesse), obedece\_ordenes(jesse), tiene\_tarea(jesse), recibe\_paga(jesse), a\_cargo\_de(jesse, n_1)\}$.
- Átomos que ya no pueden ser inferidos luego de remover a_3 : $\{toma\_decisiones(walter)\}$.
- Átomos que ya no pueden ser inferidos luego de remover a_4 : $\{toma\_decisiones(jesse)\}$.
- Átomos que ya no pueden ser inferidos luego de remover a_5 : $\{supervisa(skyler, walter), supervisor(skyler)\}$.
- Átomos que ya no pueden ser inferidos luego de remover a_6 : ninguno.

Ademas, asumamos que consideramos más plausible el átomo a_3 que el átomo a_4, y el orden para los átomos en kernels de datos es entonces $a_6 \lessdot a_4 \lessdot a_3 \lessdot a_5 \lessdot a_2$. Entonces, tenemos que

$$\varrho(\{a_2, a_4\}) = \{a_4\}$$
$$\varrho(\{a_3, a_5\}) = \{a_3\}$$
$$\varrho(\{a_3, a_6\}) = \{a_6\}$$
$$\varrho(\{a_2, a_5\}) = \{a_5\}$$

Por lo tanto, usando un operador de consolidación de ontologías Datalog$^{\pm}$ basado en Kernel Contraction como fue definido en la Definición 5.6 obtenemos la siguiente ontología coherente y consistente:

$$\Upsilon_{\rho,\varrho}(KB) = \left\{ \begin{array}{ll} D': & \{jefe(walter), supervisa(walter, jesse), \\ & a\_cargo\_de(jesse, distribution), \\ & a\_cargo\_de(walter, cooking) \\ & en\_huelga(mike)\} \\ \\ \Sigma'_{NC}: & \{obedece\_ordenes(X) \wedge toma\_desiciones(X) \rightarrow \bot, \\ & supervisa(Y, X) \wedge supervisor(X) \rightarrow \bot, \\ & ausente(X) \wedge en\_huelga(X) \rightarrow \bot\} \\ \\ \Sigma'_{E}: & \{a\_cargo\_de(X, Y) \wedge a\_cargo\_de(X, Y') \rightarrow Y = Y'\} \\ \\ \Sigma'_{T}: & \{empleado(X) \rightarrow es\_supervisado(X), \\ & es\_supervisado(X) \rightarrow obedece\_ordenes(X), \\ & jefe(X) \rightarrow obtiene\_ganancias(X), \\ & supervisa(Y, X) \rightarrow supervisor(Y), \\ & supervisa(Y, X) \rightarrow empleado(X), \\ & es\_supervisado(X) \rightarrow tiene\_tarea(X), \\ & tiene\_tarea(X) \rightarrow recibe\_paga(X), \\ & tiene\_tarea(X) \rightarrow \exists Y\, a\_cargo\_de(X, Y) \end{array} \right\}$$

5.5. Conclusiones

En este capítulo hemos presentado un enfoque que posibilita la consolidación de ontologías Datalog$^{\pm}$. Tal proceso está basado en las ideas de Hansson [Han94, Han99] de atacar conflictos mínimos conocidos como kernels, removiendo de los mismos fórmulas para resolver los conflictos. Además de atender conflictos de inconsistencia, los operadores definidos también se enfocan en el otro tipo de conflictos que pueden surgir en entornos ontológicos: la incoherencia. De esta forma puede verse a los operadores como unos que trabajan en dos fases: los mismos comienzan por resolver los problemas de incoherencia que aparezcan en la ontología mediante la remoción de TGDs, para luego dar paso a la restauración de la consistencia a través de la eliminación de átomos del componente D. En ambos casos, las selecciones se hacen a través de funciones de incisión que deciden

qué remover en base a una relación general $\preccurlyeq$, mirando para ello localmente en cada kernel en la ontología.

Para caracterizar apropiadamente el proceso de consolidación de ontologías hemos introducido un conjunto de postulados, adaptando intuiciones introducidas para otros formalismos de representación de conocimiento por Hansson [Han94], Konieczny y Pino-Pérez [KP02], y Fuhrmann [Fuh97]. A tales propiedades las hemos aumentado con los postulados de Optimidad Local para Restricciones y Optimidad Local para Datos, los cuales son utilizados para enfocarnos en aquellas consolidaciones que minimizan la pérdida de información con respecto a $\preccurlyeq$, donde la misma es considerada localmente, *esto es,* para cada conjunto minimalmente inconsistente/incoherente. El conjunto básico de Postulados (OCP1-OCP7) versa entonces en diferentes aspectos de la consolidación de ontologías, desde requerir que nada sea agregado a la ontología original hasta que aquello que no involucra conflictos sea retenido. Este conjunto básico de postulados implica a su vez ciertas otras propiedades. En particular, dos propiedades que se derivan del conjunto de postulados básicos y que son muy importantes a efectos del objetivo de los operadores de consolidación son los de **Coherencia** y **Consistencia**. Es importante remarcar que, como es tradicional en la teoría de cambios, los operadores no se encuentran atados a las construcciones particulares que se presentan en la presente tesis, sino que son un aporte general a cualquier operador de consolidación de ontologías Datalog$^{\pm}$ (y que pueden ser incluso generalizados a otros lenguajes ontológicos con relativa sencillez). Es decir, cualquier operador de consolidación de ontologías Datalog$^{\pm}$ puede ser analizado a la luz de los postulados presentados, y no solamente aquellos introducidos en este trabajo. Por lo tanto, el conjunto de postulados introducidos corresponde un marco formal general del comportamiento de operadores de consolidación.

Luego de presentar los postulados que moldean el comportamiento de los operadores de consolidación de ontologías Datalog$^{\pm}$, el el presente capítulo nos hemos dedicado a la definición de una clase particular de tales operadores, presentando para ello una construcción completa que puede generarlos. Para ello primeramente procedimos a definir que corresponden con kernels de dependencias (conjuntos insatisfacibles de TGDs mínimos bajo inclusión conjuntista) y kernels de datos (conjuntos mínimos de átomos que hacen a la ontología inconsistente). En particular, para los kernels de dependencias hemos demostrado que los mismos son independientes de la instancia particular del componente D en la ontologías, es decir que los mismos no dependen de los átomos. Esto es particular-

mente interesante porque nos da la pauta de que el conjunto de kernels de dependencias de una ontología puede ser obtenido sin considerar el componente D de la misma, lo que es de utilidad a la hora de separar la resolución de incoherencias de la resolución de inconsistencias.

Una vez que hemos identificado los kernels sobre los que aplicaremos las operaciones de contracción en pos de resolver incoherencias e inconsistencias, debemos establecer como tales problemas son resueltos. Para ello utilizamos funciones de incisión, que toman como argumento de entrada kernels de dependencias y kernels de datos y selecciona qué TGDs serán removidas de Σ_T y qué átomos serán removidos de D. Es posible definir funciones de incisión generales, que consideran al mismo tiempo tanto incoherencia como inconsistencia. Sin embargo, tal enfoque acarrea un problema:para hacer esto deberíamos calcular los kernels de dependencias y de datos y resolver los conflictos al mismo tiempo. Como se mostró el en Ejemplo 5.4, en la presencia de incoherencia esto puede no ser la mejor solución, al considerarlo a la luz de la pérdida mínima de información. Esto es generalizado por la Proposición 5.4 y por el Corolario 5.5, que establecen que para el caso de átomos relevantes a conjuntos insatisfacibles los mismos deben ser obligatoriamente eliminados por una función de incisión general, incluso cuando al mismo tiempo la función de incisión resuelve el problema de incoherencia que hacía que el átomo relevante sea un kernel (resolviendo indirectamente la inconsistencia).

Para resolver tal situación es que hemos introducido una separación entre las funciones de incisión que trabajan sobre el conjunto de TGDs de una ontología de aquellas que se encargan de la remoción de átomos de la misma. En base a tal separación definimos a los operadores de consolidación de ontologías Datalog$^{\pm}$ como aquellos que utilizan una función de incisión en restricciones sobre el componente Σ y una función de incisión en datos sobre D. Se puede considerar que tales operadores trabajan en "fases" separadas: primero se resuelven todos los problemas de incoherencia, y luego la resolución de inconsistencias se hace sobre la ontología intermedia obtenida mediante la remoción de TGDs. Es decir, por un lado, las TGDs son removidas de Σ (solamente TGDs, ya que los kernels de dependencias no contienen EGDs ni NCs). Por el otro lado, como la función de incisión en datos usa la ontología intermedia $KB^{\star}$ en lugar de KB entonces sólo átomos de D que estén en conflicto con $\Sigma \setminus \rho(KB)$ son removidos, ya que los kernels de datos son calculados en base a las restricciones obtenidas luego de aplicar sobre Σ la función de incisión en restricciones.

Por último, hemos establecido la relación entre los postulados y la construcción presentada a través de un teorema de representación, el cual establece que la relación biunívoca entre el conjunto de postulados OCP1-OCP7 y la construcción en base a funciones de incisión. La importancia de tal resultado es que determina que cualquier construcción que satisfaga tales postulados se corresponderá necesariamente con los operadores presentados en este capítulo, y por lo tanto los resultados de esos operadores pueden ser obtenidos por nuestra construcción.

Capítulo 6

Refinamiento del proceso de consolidación de ontologías Datalog$^{\pm}$

En el Capítulo 5 hemos introducido un proceso novel para la consolidación de ontologías Datalog$^{\pm}$. Tal operador posee varias propiedades que hacen que la ontología final obtenida del proceso de consolidación tenga las características deseadas de coherencia y consistencia. Sin embargo, el proceso introducido puede ser refinado cuando consideramos el aspecto de *mínima pérdida de información*, el cual es un concepto muy arraigado a la teoría de cambio de creencias. Como veremos en el presente capítulo, un operador basado en Kernel Contraction puede inducir remociones innecesarias de fórmulas. En el presente capítulo de la tesis elaboramos sobre los operadores ya introducidos para conseguir una variante optimal (respecto de pérdida de información) de los mismos, los *operadores de consolidación de ontologías Datalog$^{\pm}$ basados en Cluster Contraction*.

6.1. Necesidad de refinamiento

A continuación analizaremos el comportamiento de los operadores introducidos en el Capítulo 5 a la luz de una nueva perspectiva, la (excesiva) pérdida de información. Para esto miraremos en mayor detalle el ejemplo del comportamiento de tales operadores presentado previamente, poniendo de evidencia tal debilidad. Una vez que hayamos establecido claramente la situación que queremos evitar procederemos a proponer una nueva propiedad para los operadores de consolidación de ontologías Datalog$^{\pm}$, la que una vez satisfecha asegura que tal situación no sucede.

6.1.1. Problemas asociados a la consolidación basado en Kernel Contraction

A pesar de obtener una ontología final coherente y consistente, los operadores introducidos en el Capítulo 5 tienen una importante desventaja: incluso cuando el operador sólo selecciona una fórmula por cada kernel, bajo ciertas condiciones el operador puede remover más fórmulas de las absolutamente necesarias para obtener una ontología adecuadamente consolidada.

Para ver tal situación consideremos nuevamente el comportamiento del operador de consolidación de ontologías Datalog$^{\pm}$ basado en Kernel Contraction en el Ejemplo 5.5 de la Sección 5.4. En tal ejemplo se puede ver que el proceso de consolidación presentado tiene ciertas fallas y puede ser refinado. Centremos la atención en los kernels de datos en el ejemplo, y como los distintos problemas de consistencia son finalmente resueltos; claramente ciertas remociones efectuadas por el operador son innecesarias. Por ejemplo, considere los kernels de datos $\{a_2, a_5\}$ y $\{a_3, a_5\}$. Como en el orden entre conjuntos de átomos tenemos que $a_3 \lessdot_1 a_5 \lessdot_1 a_2$, para el primer kernel la función de incisión en datos remueve el átomo a_5, mientras que para el segundo kernel remueve a_3, lo que ciertamente es innecesario ya que la inconsistencia que surgía del segundo kernel ya está resuelta si decidimos remover a_5 para resolver el conflicto en $\{a_2, a_5\}$. Es más, el átomo a_3 se pierde en el proceso de consolidación porque el mismo no puede ser generado por la aplicación de ninguna de las TGDs remanentes al considerarlas con la base de datos D obtenida.

6.1.2. Una propiedad adicional: la mínima pérdida de información

Para paliar la situación presentada comenzaremos expandiendo el conjunto de Postulados presentados previamente con uno adicional que expresa formalmente nuestra noción de mínima pérdida de información para operadores de consolidación de ontologías Datalog$^{\pm}$, que luego servirá de guía para el desarrollo de una nueva construcción que cumpla con la misma. La propiedad que se espera sea satisfecha por un operador optimal (en el sentido de mínimo cambio) es la de **Mínima Pérdida de Información**.

- **(Mínima Pérdida de Información)**: Si $KB' \subseteq KB$ es coherente y consistente, entonces se verifica que $\Upsilon(KB) \not\subset KB'$.

> No existe una ontología coherente y consistente obtenida a partir de la ontología original que contenga estrictamente a la ontología consolidada.

La intuición detrás del postulado es que un operador de consolidación que lo satisface nos dará como resultado una ontología tal que si agregamos cualquiera de las fórmulas eliminadas entonces la KB resultante sería incoherente o inconsistente; es decir, el operador realizó una cantidad de cambios *mínima* en la resolución de conflictos.

6.2. Operador de consolidación basado en Cluster Contraction

Es claro que un operador de consolidación como el definido en la Definición 5.6 no logra satisfacer **Mínima Pérdida de Información**. Por ejemplo, considerando $\Upsilon(KB) = \{D', \Sigma'\}$ en el Ejemplo 5.5 (*esto es,* la ontología consolidada obtenida finalmente) tenemos que $\{\Upsilon(KB) \cup \{a_3\}\} \subsetneq KB$ es coherente y consistente. A su vez, claramente $\Upsilon(\Psi) \subsetneq \{\Upsilon(\Psi) \cup \{a_3\}\}$, y la pérdida de información es mayor de la necesaria en la resolución de conflictos de (en este caso) consistencia.

La razón detrás de la no minimalidad del operador presentado anteriormente es el uso de kernels de manera aislada, sin tener en cuenta aquellas ocasiones donde los conflictos minimales están en una relación de "colisión". A continuación presentamos un refinamiento del proceso de consolidación basado en Kernel Contraction introducido previamente cuyo objetivo es satisfacer el requerimiento de mínimo cambio atacando la fuente de tales problemas. Para lograr esto definimos un nuevo operador de consolidación que satisface tanto el principio de **Mínima Pérdida de Información** como las propiedades de **Coherencia** y **Consistencia** presentadas en el Capítulo 5.

6.2.1. Clusterización

Como anticipamos, la base del refinamiento propuesto al uso de kernels se basa en la detección de aquellos que se relacionan entre ellos de alguna forma. Para lograr esto, en lugar de realizar incisiones sobre kernels el nuevo operador realiza las mismas sobre *clusters*, una noción introducida en primera instancia en [MPS+07] y luego estudiada

como la base para el manejo de inconsistencias en [LMS12, MPP^{+}14]. La estructura de los clusters hace posible identificar conflictos relacionados para de esta forma encarar una resolución más global teniendo en cuenta tal relación. Los clusters son obtenidos a través de una relación de *solapamiento* (overlapping) entre kernels.

Definición 6.1 (Solapamiento, Equivalencia) *Sea $\mathcal{L}$ un lenguaje de primer orden, $\mathcal{R} \subset \mathcal{L}$ un esquema relacional y $\mathcal{L}_{\mathcal{R}}$ el sublenguaje generado por $\mathcal{R}$. Dados $A \subset \mathcal{L}_{\mathcal{R}}$ y $B \subset \mathcal{L}_{\mathcal{R}}$, decimos que los mismos se* solapan*, denotado $A\,\theta B$, si y sólo si $A \bigcap B \neq \emptyset$. Adicionalmente, dado un conjunto de conjuntos de fórmulas de primer orden $\mathcal{M} \subset 2^{\mathcal{L}_{\mathcal{R}}}$ denotamos por $\theta^{*}_{\mathcal{M}}$ la relación de equivalencia obtenida sobre $\mathcal{M}$ a través del cierre transitivo y reflexivo de θ.*

Mediante la explotación de la relación de solapamiento entre kernels de dependencias y datos podemos definir los *clusters de dependencias* y los *clusters de datos*, respectivamente.

Definición 6.2 (Clusters de dependencias) *Sea $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$, y $\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de kernels de dependencias para KB. Sea θ la relación de solapamiento, y $\mathcal{K} = \perp\!\!\!\perp_{(\Sigma,KB)}/\theta^{*}_{\perp\!\!\!\perp_{(\Sigma,KB)}}$ el conjunto cociente para la relación de equivalencia obtenida sobre $\perp\!\!\!\perp_{(\Sigma,KB)}$. Un* cluster de dependencias *es un conjunto $X = \bigcup_{Y \in [\kappa]} Y$, donde $[\kappa] \in \mathcal{K}$. Denotamos por $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de todos los clusters de dependencias de KB.*

Definimos a los *clusters de datos* de forma análoga a los clusters de dependencias.

Definición 6.3 (Clusters de datos) *Sea $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$, y $\perp\!\!\!\perp_{(D,KB)}$ el conjunto de kernels de datos para KB. Sea θ la relación de solapamiento, y $\mathcal{K} = \perp\!\!\!\perp_{(D,KB)}/\theta^{*}_{\perp\!\!\!\perp_{(D,KB)}}$ el conjunto cociente para la relación de equivalencia obtenida sobre $\perp\!\!\!\perp_{(D,KB)}$. Un* cluster de datos *es un conjunto $X = \bigcup_{Y \in [\kappa]} Y$, donde $[\kappa] \in \mathcal{K}$. Denotamos por $\perp\!\!\!\perp\!\!\!\perp_{(D,KB)}$ el conjunto de todos los clusters de datos de KB.*

Intuitivamente, un cluster de dependencias agrupa kernels de dependencias que tienen alguna TGD en común, de forma transitiva; los clusters de datos agrupan kernels de datos con átomos en común.

Ejemplo 6.1 (Clusters de dependencias y clusters de datos) *Supongamos que tenemos KB tal que* $\perp\!\!\!\perp_{(\Sigma,KB)} = \{\{\sigma_1, \sigma_2\}, \{\sigma_1, \sigma_3\}, \{\sigma_4, \sigma_5\}\}$ *y* $\perp\!\!\!\perp_{(D,KB)} = \{\{a_1, a_2\}, \{a_1, a_3\}, \{a_2, a_4, a_5\}\}$. *Entonces, tenemos dos clusters de dependencias basados en esos kernels, agrupando los primeros dos kernels en un cluster (debido a* σ_1*) y el kernel restante en otro cluster; esto es,* $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)} = \{\{\sigma_1, \sigma_2, \sigma_3\}, \{\sigma_4, \sigma_5\}\}$. *Por el otro lado, para el caso de los clusters de datos tenemos que* $\perp\!\!\!\perp\!\!\!\perp_{(D,KB)} = \{\{a_1, a_2, a_3, a_4, a_5\}\}$.

La siguiente proposición expresa que, como los clusters están basados en la utilización de clases de equivalencia, todo kernel está incluido en un y sólo un cluster.

Proposición 6.1 *Sea KB una ontología Datalog*$^{\pm}$, $\perp\!\!\!\perp_{(\Sigma,KB)}$ *el conjunto de kernels de dependencias para KB y* $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ *el conjunto de clusters de dependencias obtenido en base a* $\perp\!\!\!\perp_{(\Sigma,KB)}$, $\perp\!\!\!\perp_{(D,KB)}$ *el conjunto de kernels de datos para KB y* $\perp\!\!\!\perp\!\!\!\perp_{(D,KB)}$ *el conjunto de clusters de datos para KB obtenido en base a* $\perp\!\!\!\perp_{(D,KB)}$.

Entonces, $Y \in \perp\!\!\!\perp_{(\Sigma,KB)}$ *es tal que* $Y \subseteq X$ *para algún* $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ *si y solo si* $Y \nsubseteq X'$ *para todo* $X' \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ *tal que* $X \neq X'$. *Análogamente,* $Y \in \perp\!\!\!\perp_{(D,KB)}$ *es tal que* $Y \subseteq X$ *para algún* $X \in \perp\!\!\!\perp\!\!\!\perp_{(D,KB)}$ *si y solo si* $Y \nsubseteq X'$ *para todo* $X' \in \perp\!\!\!\perp\!\!\!\perp_{(D,KB)}$ *tal que* $X \neq X'$.

Demostración: ver Apéndice A [página 204].

Como corolario de la Proposición 6.1 tenemos que una fórmula perteneciente a algún kernel está incluída en un único cluster.

Corolario 6.2 (Corolario de la Proposición 6.1) *Sea KB una ontología Datalog*$^{\pm}$, $\perp\!\!\!\perp_{(\Sigma,KB)}$ *el conjunto de kernels de dependencias para KB y* $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ *el conjunto de clusters de dependencias obtenido en base a* $\perp\!\!\!\perp_{(\Sigma,KB)}$, $\perp\!\!\!\perp_{(D,KB)}$ *el conjunto de kernels de datos para KB y* $\perp\!\!\!\perp_{(D,KB)}$ *el conjunto de clusters de datos para KB obtenido en base a* $\perp\!\!\!\perp_{(D,KB)}$.

Sea $\alpha \in Y$ *para algún* $Y \in \perp\!\!\!\perp_{(\Sigma,KB)}$ *y* $\beta \in Y'$ *para algún* $Y' \in \perp\!\!\!\perp_{(D,KB)}$. *Entonces,* $\alpha \in X$ *para algún* $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ *si y solo si* $\alpha \notin X'$ *para todo* $X' \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ *tal que* $X \neq X'$. *Análogamente,* $\beta \in Y'$ *es tal que* $\beta \in X$ *para algún* $X \in \perp\!\!\!\perp\!\!\!\perp_{(D,KB)}$ *si y solo si* $\beta \notin X'$ *para todo* $X' \in \perp\!\!\!\perp\!\!\!\perp_{(D,KB)}$ *tal que* $X \neq X'$.

Demostración: ver Apéndice A [página 205].

El siguiente lema establece que cuando dos *KBs* son iguales las mismas tienen los mismos clusters de dependencias y datos.

Lema 6.3 *Sean KB_1 y KB_2 dos ontologías Datalog$^{\pm}$ tales que $KB_1 = KB_2$. Entonces, $\perp\!\!\!\perp_{(D,KB_1)} = \perp\!\!\!\perp_{(D,KB_2)}$ y $\perp\!\!\!\perp_{(\Sigma,KB_1)} = \perp\!\!\!\perp_{(\Sigma,KB_2)}$.*

Demostración: ver Apéndice A [página 206].

6.2.2. Funciones de incisión en clusters de dependencias y en clusters de datos

El uso de clusters en lugar de trabajar directamente sobre kernels ayuda a prevenir situaciones como la presentada en el Ejemplo 5.5, ya que usando los primeros podemos saber cuando dos o más kernels están relacionados, y entonces aplicar una solución global a los mismos para resolver los conflictos representados por los kernels. Sin embargo, el uso de clusters trae aparejado un problema: al remover alguna fórmula de un kernel el conjunto resultante siempre es coherente/consistente ya que los kernels son conjuntos mínimos, pero en el caso de los clusters no podemos asegurar que la remoción de una fórmula nos de como resultado un conjunto con todos los conflictos agrupados resueltos [LMS12]. Por esta razón no es aconsejable directamente reutilizar las funciones de incisiones definidas en las Definiciones 5.4 y 5.5, como se muestra a continuación.

Ejemplo 6.2 *Considere el cluster de dependencias $\{\sigma_1, \sigma_2, \sigma_3\}$ en $\perp\!\!\!\perp_{(\Sigma,KB)}$, el cual agrupa los kernels de dependencias relacionados $\{\sigma_1, \sigma_2\}$ y $\{\sigma_1, \sigma_3\}$. Supongamos que tenemos un orden entre restricciones tal que $\sigma_2 \lessdot \sigma_3 \lessdot \sigma_1$, y una función de incisión en restricciones ρ definida en base a $\lessdot$. Entonces, tenemos que ρ es tal que $\rho(\{\sigma_1, \sigma_2, \sigma_3\}) = \sigma_2$.*

Claramente ρ es una función de incisión válida (tomando clusters como argumentos) de acuerdo a la Definición 5.4, ya que cumple con todas las condiciones en la definición. Sin embargo, es evidente que remover σ_2 no es suficiente para resolver todos los conflictos de coherencia, ya que entonces el conjunto insatisfacible $\{\sigma_1, \sigma_3\}$ seguiría siendo parte de $\Upsilon(KB)$.

A continuación introducimos una nueva clase de funciones de incisión adaptadas a las peculiaridades de los clusters.

Definición 6.4 (Función de incisión en clusters de dependencias) *Sea $\mathcal{R}$ un esquema relacional y $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ sobre $\mathcal{R}$ donde $\Sigma = \Sigma_T \cup \Sigma_E \cup$*

Σ_{NC}. *Sea* $\perp\!\!\!\perp_{(\Sigma,KB)}$ *el conjunto de kernels de dependencias para KB y* $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ *el conjunto de clusters de dependencias para KB obtenido en base a* $\perp\!\!\!\perp_{(\Sigma,KB)}$. *Una* función de incisión en clusters de dependencias *para KB es una función* $\phi : 2^{2^{\mathcal{L}_\mathcal{R}}} \longrightarrow 2^{\mathcal{L}_\mathcal{R}}$ *tal que todas las siguientes condiciones se verifican:*

- $\phi(KB) \subseteq \bigcup(\perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)})$.
- *Para todo* $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ *e* $Y \in \perp\!\!\!\perp_{(\Sigma,KB)}$ *tal que* $Y \subseteq X$ *se verifica que* $(Y \cap \phi(KB)) \neq \emptyset$.
- *Para todo* $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ *se verifica que* $T = (X \cap \phi(KB))$ *es tal que no existe* $R \subset X$ *donde* R *satisface las dos condiciones previas y* $R \subsetneq T$.

Definición 6.5 (Función de incisión en clusters de datos) *Sea* $\mathcal{R}$ *un esquema relacional y* $KB = (D, \Sigma)$ *una ontología Datalog*$^{\pm}$ *sobre* $\mathcal{R}$ *donde* $\Sigma = \Sigma_{T} \cup \Sigma_{E} \cup \Sigma_{NC}$. *Sea* $\perp\!\!\!\perp_{(D,KB)}$ *el conjunto de kernels de datos para KB y* $\perp\!\!\!\perp\!\!\!\perp_{(D,KB)}$ *el conjunto de clusters de datos para KB obtenido en base a* $\perp\!\!\!\perp_{(D,KB)}$. *Una* función de incisión en clusters de datos *para KB es una función* $\varphi : 2^{2^{\mathcal{L}_\mathcal{R}}} \longrightarrow 2^{\mathcal{L}_\mathcal{R}}$ *tal que todas las siguientes condiciones se verifican:*

- $\varphi(KB) \subseteq \bigcup(\perp\!\!\!\perp\!\!\!\perp_{(D,KB)})$.
- *Para todo* $X \in \perp\!\!\!\perp\!\!\!\perp_{(D,KB)}$ *e* $Y \in \perp\!\!\!\perp_{(D,KB)}$ *tal que* $Y \subseteq X$ *se verifica que* $(Y \cap \varphi(KB)) \neq \emptyset$.
- *Para todo* $X \in \perp\!\!\!\perp\!\!\!\perp_{(D,KB)}$ *se verifica que* $T = (X \cap \varphi(KB))$ *es tal que no existe* $R \subset X$ *donde* R *satisface las dos condiciones previas y* $R \subsetneq T$.

Es importante notar que tanto la Definición 6.4 como la Definición 6.5 se aseguran de que no existan problemas debido a la no minimalidad de los clusters (*esto es,* como el mostrado en el Ejemplo 6.2), ya que aseguran que la intersección de las funciones con los kernels es no vacía. Otra característica destacable de las funciones de incisión en clusters es que la elección del subconjunto del cluster que se removerá para restaurar la coherencia/consistencia en los conflictos que el cluster agrupa es óptima para el mismo, en el sentido de que ningún subconjunto de lo elegido puede resolver efectivamente los conflictos agrupados (y por lo tanto es una solución óptima global para los kernels incluidos en éste).

6.2.3. Consolidación de ontologías basada en Cluster Contraction

Una vez que tenemos las estructuras que agruparan conflictos mínimos relacionados (clusters) y funciones de incisión adecuadas para trabajar sobre las mismas, procedemos a definir un nuevo operador de consolidación de ontologías Datalog$^{\pm}$, el *operador de consolidación de ontologías Datalog$^{\pm}$ basado en Cluster Contraction*, el cual tiene como objetivo principal el evitar realizar remociones innecesarias de fórmulas. Nuevamente, como en el caso del operador basado en Kernel Contraction, el proceso se inicia con la restauración de la coherencia del componente Σ. Para ello, se obtienen los clusters de dependencias en base a los kernels de dependencias en $\perp\!\!\!\perp_{(\Sigma,KB)}$. Luego, la restauración de consistencia del componente D se hace utilizando los clusters de datos que podemos encontrar al analizar la relación de solapamiento entre los kernels de datos obtenidos usando el resultado de aplicar una función de incisión en clusters de dependencias sobre Σ. Formalmente, el operador de consolidación de ontologías Datalog$^{\pm}$ basado en la contracción de clusters es definido de la siguiente forma.

Definición 6.6 (Operador de consolidación basado en Cluster Contraction) *Sea KB una ontología Datalog$^{\pm}$, ϕ una función de incisión en clusters de dependencias y φ una función de incisión en clusters de datos. Además, sea $KB^{\star} = (D, \Sigma \setminus \phi(KB))$ la ontología resultante de remover de KB aquellas TGDs seleccionadas por ϕ. El operador de consolidación basado en Cluster Contraction $\Psi_{\phi,\varphi}(KB)$, es definido como:*

$$\Psi_{\phi,\varphi}(KB) = (D \setminus \varphi(KB^{\star}), \Sigma \setminus \phi(KB))$$

Al igual que en el caso del operador basado en Kernel Contraction, un operador definido como en la Definición 6.6 solo remueve átomos y TGDs, dejando intactos los componentes Σ_{E} y Σ_{NC} de la ontología original KB.

6.3. Relación entre los operadores basados en Kernel Contraction y los basados en Cluster Contraction

En el presente capítulo hemos introducido un operador alternativo a aquél presentado en el Capítulo 5 para conseguir la consolidación de ontologías Datalog$^{\pm}$. A continuación nos enfocaremos en analizar la relación existente entre ambos enfoques.

Primero mostramos que, como se puede inferir del Ejemplo 6.2, una función de incisión en kernels *no es* una función de incisión en clusters, y por lo tanto un operador definido como en la Definición 5.6 no es un operador definido como en la Definición 6.6. Para ver mejor esto consideremos el siguiente ejemplo.

Ejemplo 6.3 *Consideremos nuevamente la incisión en el Ejemplo 6.2. Claramente, si bien ρ es una función de incisión de restricciones válida para kernels acorde con la Definición 5.4, no es una función de incisión en clusters de dependencias de acuerdo a la Definición 6.4 ya que para $\{\sigma_1, \sigma_3\} \in \perp\!\!\!\perp_{(\Sigma,KB)}$ tenemos que $\rho \cap \{\sigma_1, \sigma_3\} = \emptyset$, y por lo tanto la segunda condición en la Definición 6.4 no es respetada por ρ.*

Una situación análoga se da cuando analizamos las funciones de incisión para datos.

A continuación analizaremos el otro sentido posible de la inclusión, es decir si se verifica o no que todo operador sobre clusters es en efecto un operador sobre kernels. De acuerdo con el Teorema 5.1, para ello basta con verificar que los operadores de cluster cumplen con los postulados de **Inclusión**, **Éxito para Restricciones**, **Éxito para Datos**, **Vacuidad para Restricciones**, **Vacuidad para Datos**, **Optimalidad Local para Restricciones** y **Optimalidad Local para Datos**. En el siguiente ejemplo se muestra como estos últimos postulados no son satisfechos por operadores definidos como en la Definición 6.6, y por lo tanto podemos afirmar que los mismos no se corresponden con operadores definidos como en la Definición 5.6.

Ejemplo 6.4 *Nuevamente, continuemos con KB, $\perp\!\!\!\perp_{(\Sigma,KB)}$ y $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ del Ejemplo 6.2. Una de las posibles incisiones que ϕ puede realizar en $\{\sigma_1, \sigma_2, \sigma_3\} \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$. es $\phi(KB) = \sigma_1$, ya que es el menor subconjunto del kernel tal que la intersección con cada kernel es*

no vacía, y claramente el único subconjunto estricto es $\emptyset$*, quien no tiene intersección con ningún kernel. Por lo tanto, un operador* $\Psi_{\phi,\varphi}(KB)$ *podría remover* σ_1*.*

Analicemos ese comportamiento a la luz del postulado de ***Optimalidad Local para Restricciones****. Consideremos el kernel* $\{\sigma_1, \sigma_2\} \in \perp\!\!\!\perp_{(\Sigma,KB)}$*. Tenemos que* $\sigma_1 \in \Sigma$ *es tal que* $\sigma_1 \notin \Psi_{\phi,\varphi}(KB)$*, y por lo tanto* $\sigma_1 \notin \Sigma'$*.*

Un posible μ *tal que* μ *es satisfacible y* $\mu \cup \sigma_1$ *es insatisfacible es* $\mu = \{\sigma_2\}$*. Claramente, para este* μ *no vale* ***Optimalidad Local para Restricciones****, ya que* $\sigma_2 \lessdot \sigma_1$*. El otro* μ *posible es* $\mu = \{\sigma_3\}$*. Nuevamente, tenemos que la elección en tal kernel no es óptima ya que* $\sigma_3 \lessdot \sigma_1$*. Por lo tanto, no existe* μ *tal que* μ *es satisfacible y* $\mu \cup \sigma_1$ *es insatisfacible donde se dé que* $\sigma_1 \lessdot \beta$ *para todo* $\beta \in \mu$*, y por lo tanto no se cumple* ***Optimalidad Local para Restricciones****.*

Un ejemplo análogo puede encontrarse para mostrar que los operadores de consolidación basados en Cluster Contraction no satisfacen ***Optimalidad Local para Datos****.*

Es decir que tenemos que ni los operadores basados en Kernel Contraction son una subclase de los operadores basados en Cluster Contraction ni viceversa. Esto es, los enfoques locales y globales de resolución de inconsistencias en ontologías Datalog$^{\pm}$ basados en pérdida de información pueden llegar a soluciones distintas a la hora de obtener conocimiento consolidado, inclusive para el caso donde el orden en el cual las incisiones se basan es el mismo.

6.4. Relación entre construcción y postulados - Teorema de Representación

Finalmente, ahondamos en la relación existente entre la construcción del operador de consolidación basado en Cluster Contraction y las propiedades presentadas tanto en el Capítulo 5 como en el presente. Para ello, introducimos el teorema de representación para un operador de consolidación de ontologías Datalog$^{\pm}$ basado en Cluster Contraction.

Teorema 6.1 (Teorema de Representación) *El operador* $\Psi_{\phi,\varphi}$ *es un* operador de consolidación de ontologías Datalog$^{\pm}$ basado en Cluster Contraction *para una ontología Datalog*$^{\pm}$ *KB si y sólo si satisface* ***Inclusión****,* ***Éxito para Restricciones****,* ***Éxito para***

***Datos*, *Vacuidad para Restricciones*, *Vacuidad para Datos* y *Mínima Pérdida de Información*.**

Demostración: ver Apéndice A [página 206].

Nuevamente, no requerimos explícitamente que $\Psi_{\phi,\varphi}$ de como resultado una ontología coherente y consistente, pero como en el caso del operador $\Upsilon_{\rho,\varrho}$ estas propiedades surgen de aquellas en el Teorema 6.1.

Corolario 6.4 (Corolario del Teorema 6.1) *$\Psi_{\phi,\varphi}$ satisface **Coherencia** y **Consistencia**.*

Demostración: ver Apéndice A [página 212].

Por lo tanto, el proceso de consolidación basado en la contracción de clusters introducido en la Definición 6.6 obtiene como resultado la ontología coherente y consistente que es la más cercana, en el sentido de pérdida de información, a la ontología original.

6.5. Ejemplo de aplicación del operador

Ejemplo 6.5 (Consolidación de ontologías Datalog$^{\pm}$ basada en Cluster Contraction) Considere nuevamente el Ejemplo 5.5. Basándonos en los kernels de dependencias obtenemos los clusters de dependencias para KB

$$\perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)} = \{\{\sigma_2, \sigma_6\}, \{\sigma_{10}\}\}$$

Nótese que, como no hay solapamiento entre los kernels de dependencias, tenemos que $\perp\!\!\!\perp_{(\Sigma,KB)} = \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$. A continuación, utilizamos la función de incisión en clusters de dependencias para resolver los problemas de incoherencia. Nuevamente, para este ejemplo utilizaremos la instancia particular de la relación $\lessdot$ introducida en el Ejemplo 5.5 que considera la cantidad de átomos perdidos y la plausibilidad asociada a las fórmulas, extendida para su uso en conjuntos de fórmulas. Es decir, asumiremos que la construcción particular de la función de incisión en clusters elige, dentro de todas aquellas incisiones que satisfacen **Mínima Pérdida de Información**, aquella preferida en base a la relación $\lessdot$ particular que hemos introducido previamente. Comenzamos presentando aquellos átomos que se pierden al eliminar los distintos subconjuntos de TGDs que eliminan TGDs involucradas de forma tal que las dos condiciones primeras en la Definición 6.4 son satisfechas:

- Átomos que ya no pueden ser inferidos luego de remover σ_2 : $obedece\_ordenes(walter), obedece\_ordenes(jesse)\}$.
- Átomos que ya no pueden ser inferidos luego de remover σ_6 : ninguno.
- Átomos que ya no pueden ser inferidos luego de remover σ_{10} : $\{ausente(mike)\}$.

Por lo tanto el orden para estas TGDs es $\sigma_6 \lessdot \sigma_{10} \lessdot \sigma_2$. En base al orden presentado, el comportamiento de la función de incisión en clusters de dependencias es el siguiente:

$$\phi(\{\sigma_2, \sigma_6\}) = \{\sigma_6\}$$
$$\phi(\{\sigma_{10}\}) = \{\sigma_{10}\}$$

Continuamos con la segunda parte del proceso de consolidación: la resolución de inconsistencias. Al igual que el caso de los operadores basados en Kernel Contraction, para esta parte el operador considera solamente TGDs que efectivamente pertenecerán a la ontología final consolidada. Para el ejemplo en cuestión esto es $\Sigma'_{_T} = \Sigma_{_T} \setminus \{\sigma_6, \sigma_{10}\}$. A partir de aquí sea $KB^\star = (D, \Sigma')$; basado en $KB^\star$ calculamos los kernels de datos.

$$\perp\!\!\!\perp_{(D,KB^\star)} = \{\{a_2, a_4\}, \{a_3, a_5\}, \{a_3, a_6\}, \{a_2, a_5\}\}$$

Basados en los kernels obtenemos los clusters de datos.

$$\perp\!\!\!\perp\!\!\!\perp_{(D,KB^\star)} = \{\{a_2, a_3, a_4, a_5, a_6\}\}$$

Ahora, para resolver los problemas de inconsistencia necesitamos calcular el orden entre los subconjuntos del cluster tales que la intersección con cada uno de los clusters es no vacía (ya que remover un sólo átomo del cluster no será suficiente), utilizando $\Sigma'_{_T}$ en lugar de $\Sigma_{_T}$. Por razones de legibilidad sólo mostraremos el orden para aquellos subconjuntos que resuelven las inconsistencias con una cantidad mínima de cambios, ya que son los únicos candidatos viables para selección por parte de las funciones de incisión en clusters de datos. Esto es, si mostramos el order para el conjunto $\{a_2, a_3\}$ entonces omitiremos $\{a_2, a_3, a_5\}$ ya que al ser un superconjunto del primero la pérdida de información sera siempre mayor o igual, así como $\{a_2\}$ y $\{a_3\}$ ya que los mismos no son capaces de resolver la inconsistencia por no tener intersección con todos los kernels incluidos en el cluster. Entonces, las distancias para los conjuntos de átomos son:

- Átomos que ya no pueden ser inferidos luego de remover $\{a_2, a_3\})$: $\{supervisa(walter, jesse), supervisor(walter), empleado(jesse), es\_supervisado(jesse), obedece\_ordenes(jesse), tiene\_tarea(jesse), recibe\_paga(jesse), a\_cargo\_de(jesse, n_1), toma_decisiones(walter)\}$.
- Átomos que ya no pueden ser inferidos luego de remover $\{a_2, a_5, a_6\}$: $\{supervisa(walter, jesse), supervisor(walter), empleado(jesse), es\_supervisado(jesse), obedece\_ordenes(jesse), tiene\_tarea(jesse), recibe\_paga(jesse), a\_cargo\_de(jesse, n_1), supervisa(skyler, walter), supervisor(skyler)\}$.
- Átomos que ya no pueden ser inferidos luego de remover $\{a_2, a_5, a_6\}$: $\{toma\_decisiones(walter), toma\_decisiones(jesse), supervisa(skyler, walter), supervisor(skyler)\}$.
- Átomos que ya no pueden ser inferidos luego de remover $\{a_4, a_5, a_6\}$: $\{toma\_decisiones(jesse), supervisa(skyler, walter), supervisor(skyler)\}$.

Dada esta información tenemos que el orden para los conjuntos de átomos introducidos es $\{a_4, a_5, a_6\} \lessdot_1 \{a_3, a_4, a_5\} \lessdot_1 \{a_2, a_3\} \lessdot_1 \{a_2, a_5, a_6\}$. Entonces, tenemos que

$$\varphi(\{\{a_2, a_3, a_4, a_5, a_6\}\}) = \{a_4, a_5, a_6\}$$

Es interesante notar cómo la noción de mínimo cambio no siempre refleja la noción de mínima pérdida de información. Para el ejemplo presentado, si elegimos eliminar el conjunto $\{a_2, a_3\}$ estaríamos eliminando una menor cantidad de átomos de KB que si elegimos el conjunto $\{a_4, a_5, a_6\}$ elegido por el operador de consolidación, pero sin embargo terminaríamos perdiendo más información ya que menos átomos pueden ser inferidos de la ontología resultante.

Entonces, usando un operador de consolidación de ontologías basado en la contracción de clusters como el introducido en la Definición 6.6 podemos obtener la siguiente ontología coherente y consistente, que es un superconjunto de la obtenida mediante la utilización de un enfoque de tratamiento local de kernels basado en Kernel Contraction:

$$\Psi_{\phi,\varphi}(KB) = \left\{ \begin{array}{ll} D': & \{jefe(walter), supervisa(walter, jesse), \\ & toma\_desiciones(walter), a\_cargo\_de(jesse, distribution), \\ & a\_cargo\_de(walter, cooking) \\ & en\_huelga(mike)\} \\ \\ \Sigma'_{NC}: & \{obedece\_ordenes(X) \wedge toma\_desiciones(X) \rightarrow \bot, \\ & supervisa(Y, X) \wedge supervisor(X) \rightarrow \bot, \\ & ausente(X) \wedge en\_huelga(X) \rightarrow \bot\} \\ \\ \Sigma'_{E}: & \{a\_cargo\_de(X, Y) \wedge a\_cargo\_de(X, Y') \rightarrow Y = Y'\} \\ \\ \Sigma'_{T}: & \{empleado(X) \rightarrow es\_supervisado(X), \\ & es\_supervisado(X) \rightarrow obedece\_ordenes(X), \\ & jefe(X) \rightarrow obtiene\_ganancias(X), \\ & supervisa(Y, X) \rightarrow supervisor(Y), \\ & supervisa(Y, X) \rightarrow empleado(X), \\ & es\_supervisado(X) \rightarrow tiene\_tarea(X), \\ & tiene\_tarea(X) \rightarrow recibe\_paga(X), \\ & tiene\_tarea(X) \rightarrow \exists Y a\_cargo\_de(X, Y) \end{array} \right\}$$

6.6. Conclusiones

En el presente capítulo hemos presentado una clase de operadores de consolidación de ontologías que modifican el comportamiento de aquellos presentados en el Capítulo 5. La diferencia entre los enfoques se da en que, mientras que los operadores presentados previamente atacan localmente a cada conflicto mínimo, en el enfoque presentado en este capítulo se utiliza una mirada más global, considerando la relación (cuando la misma existe) entre los distintos kernels. La consideración de tal relación se hace mediante el agrupamiento de kernels relacionados en una superestructura denominada cluster, la cual es obtenida explotando una relación de overlapping entre conflictos.

Al comienzo del capítulo hemos analizado el comportamiento de los operadores presentados en el Capítulo 5, enfocando el análisis en la pérdida innecesaria de información que los mismos pueden sufrir bajo ciertas condiciones, a saber la existencia de kernels relacionados y estrategias de resolución de conflictos donde la elección óptima local en cada kernel hace que debamos de remover de un kernel una fórmula (la indicada por la relación de preferencia) incluso cuando otra fórmula del mismo kernel ya ha sido seleccionada para remoción como estrategia de solución del conflicto expresado por un kernel diferente. Esta situación pone en evidencia que el conjunto de postulados introducidos previamente no es suficiente para paliar completamente el problema de pérdida mínima de información, por lo que hemos introducido una propiedad adicional. La propiedad que se espera sea satisfecha por un operador optimal (en el sentido de mínimo cambio) es la de **Mínima Pérdida de Información**. La intuición detrás del postulado es que un operador de consolidación que lo satisface nos dará como resultado una ontología tal que si agregamos cualquiera de las fórmulas eliminadas entonces la KB resultante seria incoherente o inconsistente; es decir, el operador realizó una cantidad de cambios *mínima* en la resolución de conflictos.

La propiedad de mínima pérdida no es satisfecha por los operadores definidos en el Capítulo 5, debido al uso de kernels de manera aislada, sin tener en cuenta aquellas ocasiones donde los conflictos minimales están en una relación de "colisión". Es por esto que en el presente capítulo introducimos una alternativa que se basa en la detección de la relación entre kernels, para de esta forma encontrar la solución que implique la menor pérdida posible. Para lograr esto, en lugar de realizar incisiones sobre kernels el nuevo operador realiza las mismas sobre *clusters*. La estructura de los clusters hace posible identificar conflictos relacionados a través de una relación de *solapamiento* (overlapping) entre kernels.

Sin embargo, el uso de clusters trae aparejado un problema: al remover alguna fórmula de un kernel el conjunto resultante siempre es coherente/consistente ya que los kernels son conjuntos mínimos, pero en el caso de los clusters no podemos asegurar que la remoción de una formula nos de como resultado un conjunto con todos los conflictos agrupados resueltos [LMS12]. Para paliar este problema hemos presentado una nueva clase de funciones de incisión que aseguran que una vez que el conjunto de fórmulas seleccionado por la función es removido todos los conflictos mínimos incluidos en el cluster quedan resueltos. En base a tales funciones de incisión hemos definido entonces a los operadores de consolidación de ontologías Datalog$^{\pm}$ basados en Cluster Contraction los cuales, al

igual que los operadores locales, utilizan remoción de átomos y TGDs como estrategia de resolución de incoherencias e inconsistencias.

Una vez presentada la construcción propuesta para los operadores de consolidación de ontologías basados en Cluster Contraction hemos ahondado en la relación existente entre tal construcción y las propiedades presentadas tanto en el Capítulo 5 como en el presente, a través de un teorema de representación. Tal teorema muestra que estos operadores efectivamente satisfacen el postulado de mínima pérdida de información, y por lo tanto representación soluciones óptimas desde tal punto de vista. Es decir, que los operadores basados en Cluster Contraction son tales que remueven lo mínimo que es requerido para solucionar los conflictos, y por lo tanto si fuéramos a dejar siquiera una fórmula de aquellas removidas por el operador entonces la ontología seguiría siendo incoherente/inconsistente.

A su vez, de una confrontación entre el Teorema 5.1 y el Teorema 6.1 se desprende que ambos operadores satisfacen los postulados (OCP1-OCP5). Esto es, para ir de un enfoque a otro simplemente podemos cambiar aquella propiedad que queremos satisfacer, y de esta forma podemos movernos de operadores locales a globales. La tabla en la Figura 6.1 resume la satisfacción de los enfoques con respecto a los postulados en juego.

Postulado\Enfoque	Local – Kernel Contraction	Global – Cluster Contraction
Inclusión	✓	✓
Éxito	✓	✓
Vacuidad	✓	✓
Optimalidad Local	✓	✗
Mínima Pérdida	✗	✓

Figura 6.1: Relación entre los enfoques y los postulados

De esto podemos deducir que, si existen ocasiones donde el tratamiento local de los conflictos es suficiente para obtener la mejor consolidación entonces los enfoques coincidirán para tales casos. En base a la satisfacción de los postulados tenemos la relación entre enfoques mostrada en la Figura 6.2. Nótese como en la misma se refleja aquello expresado previamente: existen casos donde los enfoques coinciden. Un claro caso (trivial) de ésta situación se da al tratar con *KBs* coherentes y consistentes. Sin embargo, no es el único caso donde tal cosa puede suceder. Otro caso que llevaría a lo mismo es aquél donde los conjuntos de kernels y de clusters coinciden, *esto es,* no hay overlapping entre

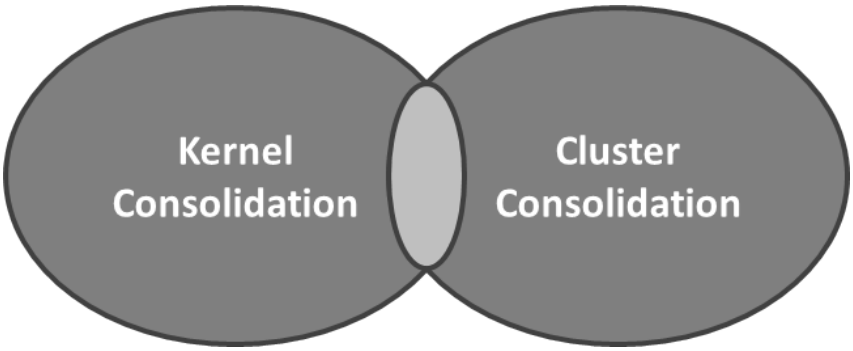

Figura 6.2: Relación entre los enfoques y los postulados

conflictos mínimos; nuevamente, para tales casos los enfoques coincidirían. Esto, si bien puede parecer trivial y no reportar mucho interés en una primera mirada, se vuelve importante a la hora de analizar a los operadores desde un punto de vista computacional. Claramente, el manejo de kernels es más sencillo que el manejo de clusters; por ejemplo, como remover una fórmula alcanza para solucionar conflictos mínimos entonces podemos comparar conjuntos unitarios de fórmulas, sin necesidad de mirar en todos los subconjuntos como sucede en Cluster Contraction. Por lo tanto, si tenemos que el conjunto de clusters coincide con el de kernels (lo que sólo involucraría una comparación de conjuntos adicional) podríamos directamente tomar el enfoque local, consiguiendo una mejor performance computacional. Es más, esto puede ser llevado un paso más allá, definiendo operadores mixtos: podríamos utilizar un enfoque local para el caso donde *un* cluster coincide con algún kernel, y utilizar un enfoque global sobre aquellos que involucran dos o más conflictos mínimos.

Capítulo 7

Aplicaciones y Trabajo Futuro

A través de los diferentes capítulos de la presente disertación hemos visto los distintos conceptos involucrados en la definición de procesos que nos permitan resolver problemas de coherencia o consistencia en ontologías Datalog$^{\pm}$. Mediante la utilización de tales procesos podemos obtener nuevas ontologías a partir de las originales, donde las ontologías resultantes puedan ser explotadas por procesos de resolución de consultas *clásicos*; es decir, aquellos que adhieren a las semánticas de las inferencias sobre lógicas de primer orden (y que por lo tanto no resultan adecuadas cuando conflictos aparecen, ya que pueden derivar todo el lenguaje): la gran ventaja de aplicar estos procesos de resolución clásicos es que ya han sido inmensamente probados a través de las años, y muchas aplicaciones han sido desarrolladas, demostrando su utilidad en el mundo real.

De la mano de lo expresado es que en el presente capítulo exploraremos diversas opciones para la aplicación de los operadores desarrollados en esta tesis. Dentro de las opciones presentadas una particularmente interesante es el uso de los operadores definidos en entornos de Bases de Datos Relacionales, lo que puede servir como un proceso de apoyo para la consecución de un objetivo más grande dentro de la línea de investigación en la cual se inserta el presente trabajo: la definición de procesos de argumentación masiva sobre bases de datos federadas. Los procesos de consolidación aquí presentados pueden efectivamente servir de apoyo a tales sistemas, ya que la integración de diferentes bases de datos puede realizarse mediante una representación de las distintas bases de datos a través de ontologías Datalog$^{\pm}$ primeramente, para luego combinar todas las ontologías mediante unión conjuntista y finalmente aplicar operadores de consolidación en la unión para resolver los conflictos que pueda surgir. La ontología Datalog$^{\pm}$ final obtenida de la

consolidación puede entonces verse como una vista unificada de todas las bases de datos involucradas en el sistema.

Finalmente, para concluir el presente capítulo versamos acerca de líneas de investigación alternativas que han surgido a partir de este trabajo, explicando lo que se espera de las mismas y cómo se piensa encararlas. Tales líneas van desde la modificación de aspectos relacionados con las construcciones propuestas (manejo de conjuntos maximalmente consistentes, particularizaciones de la relación de preferencia entre fórmulas) hasta modificaciones de más alto nivel (como el manejo de incoherencias en base al uso de debilitación de reglas en lugar de remoción).

7.1. Obtención de vistas unificadas de varias bases de datos relacionales

En los tiempos recientes cada vez se ha hecho más fuerte la necesidad de contar con métodos automáticos de resolución de conflictos, principalmente desde el surgimiento y la proliferación de entornos colaborativos donde el conocimiento proveniente de diferentes fuentes es explotado de manera conjunta. Esta interacción entre sistemas es una de las principales razones para la aparición de problemas de coherencia y consistencia: es muy común el caso donde una aplicación pretende acceder a (o utilizar de alguna forma) la fuente de datos de otra pero se encuentra con que hay problemas a nivel de esquema (*v.g.,* no sabe a que campo acceder para obtener el dato que busca) o a nivel de datos (incompletitud, inconsistencias con respecto a sus propios datos y restricciones de integridad).

Para tales casos puede ser útil el tener una sola vista que unifique de alguna manera las diversas fuentes de datos disponibles, de forma tal que el acceso a los datos se realice a través de la misma. En tales casos, una opción viable es resolver los conflictos a nivel de vista, es decir, dejar las bases de datos originales sin modificación (mejorando de esta forma la compatibilidad con sistemas legados) y resolver los conflictos en el momento de la creación de la vista (*esto es,* la federación de las bases de datos) a la que accederán los distintos sistemas implicados. Es en esta tarea donde los operadores desarrollados en la presente tesis pueden jugar un papel preponderante, permitiendo detectar y resolver los conflictos en la unión de las diversas fuentes, para de esta forma asegurarse que los mismos no son trasladados a la vista unificada, de forma tal que esta pueda ser explotada mediante

relaciones de inferencia clásica (amén de otros métodos tolerantes a tales conflictos, que claramente pueden igualmente ser aplicados), incrementando la eficiencia del sistema final.

En particular, un objetivo de la línea de investigación en la cual se enmarca la presente tesis es la definición de procesos de argumentación masiva, donde las fuentes de datos sean bases de datos relacionales. De la mano de lo anteriormente expresado, para la definición de estos sistemas que realizarán argumentación masiva en base a tales datos en lugar de modificar las bases de datos originales se procederá a integrar las mismas en una vista única. Esto permitirá la definición de nuevas arquitecturas de Sistemas de Soporte a las Decisiones (Decision Support Systems - DSS), Sistemas de Recomendación (Recommender Systems - RS) o otros sistemas expertos (*v.g.,* sistemas de Diagnóstico Médico Asistido por Computadores, Computer Aided Diagnosis - CAD) que integren bases de datos existentes para aumentar la información disponible a la hora de resolver consultas. El esquema general de estos sistemas es mostrado en la Figura 7.1.

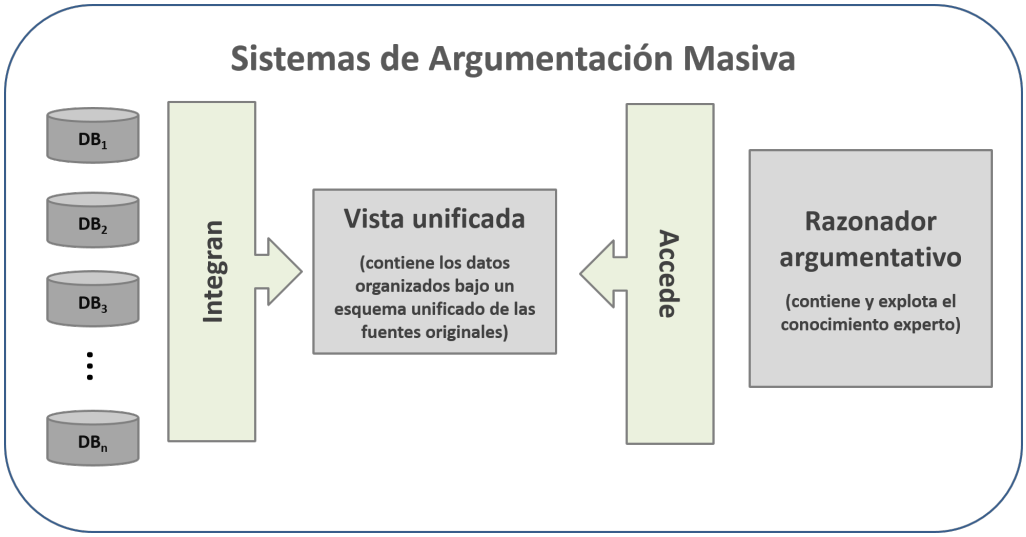

Figura 7.1: Arquitectura general para Sistemas basados en Argumentación Masiva

En particular, como veremos más adelante los operadores de consolidación de ontologías Datalog$^{\pm}$ (o algún refinamiento particular de los mismos, *v.g.,* aquellos que sólo resuelven problemas de coherencia) pueden ser aplicados en la consecución de la vista única, mediante un proceso de transformación de las bases de datos relacionales a ontologías Datalog$^{\pm}$ primeramente, para luego consolidar (la unión de) las mismas y obtener la vista. Este proceso se muestra en la Figura 7.2.

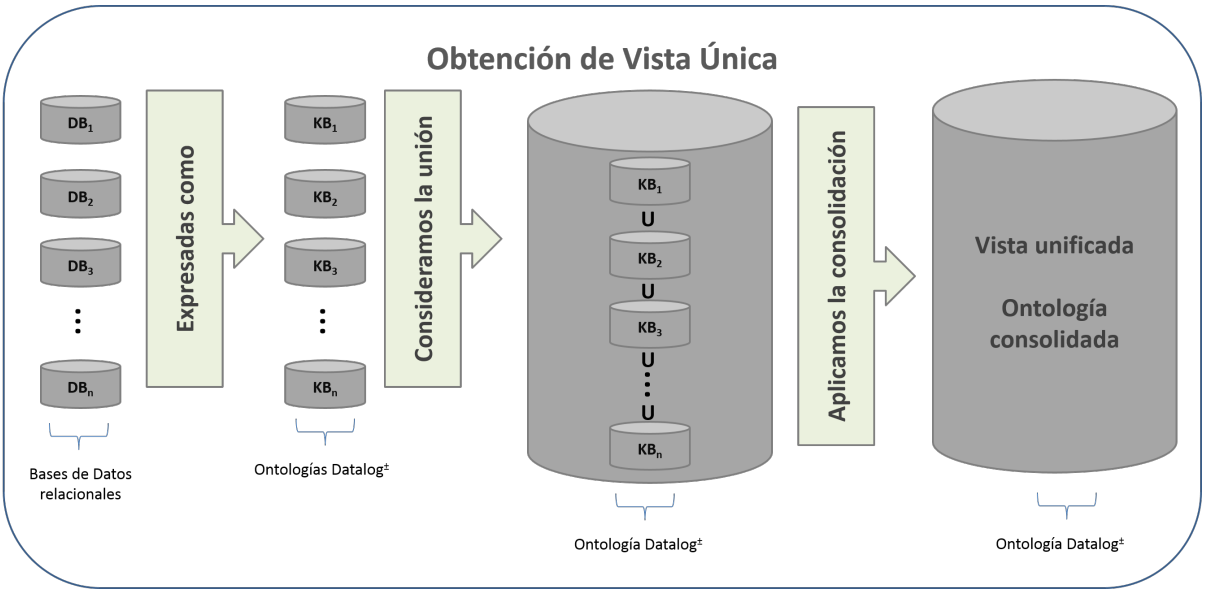

Figura 7.2: Utilización de consolidación en la obtención de la Vista Unificada

A continuación mostramos en más detalle como tales procesos pueden ser llevados a cabo. Primeramente explicamos la diferencia entre la técnica de federación aquí propuesta con respecto a otra muy conocida dentro del mundo de las bases de datos, data exchange (la cual también puede ser soportada por los operadores aquí desarrollados). Luego, procedemos a mostrar cómo pueden expresarse bases de datos relacionales a través de ontologías Datalog$^{\pm}$, lo cual es el paso previo a la utilización de una ontología unificada como federación de las bases de datos relacionales.

7.1.1. Data Federation vs. Data Exchange

La *federación de datos* (data federation, también conocida como *integración de datos - data integration* en la comunidad de bases de datos) y el *intercambio de datos* (data exchange, denominado alternativamente como *traducción de datos - data translation*) son dos problemas tradicionales en el áera de bases de datos relacionales, los cuales son reconocidos como tareas dificultosas y que suelen demandar mucho tiempo [BH08, DHI12].

Una tarea primordial a la hora de combinar información proveniente de diferentes fuentes, sin importar si tal tarea es llevada a cabo mediante federación o intercambio, es la de definir de forma precisa los mapeos entre los esquemas origen (source schemas) y

los esquemas objetivo (target schemas). La definición de tales mapeos no es trivial, y es frecuente que se involucre esfuerzo humano en esta tarea.

Una vez que tales mapeos se encuentran definidos, se procede a la utilización de los mismos para la combinación de la información. Como hemos dicho, hay dos enfoques principales, aunque relacionados.

- **Data Federation**: En la integración de datos el objetivo es brindar el acceso a los múltiples repositorios de datos mediante la utilización de una capa intermedia (la vista) la cual es unificada en base a la información obtenida de los repositorios, donde las consultas van dirigida a la vista (por lo que las aplicaciones sólo deben conocer el esquema de la misma, sin importarle los esquemas subyacentes de los repositorios reales). En su forma más tradicional, la federación no involucra una transformación de los datos, ya que se basa en la obtención de datos de las fuentes originales y la manipulación de los mismos para la resolución de consultas.
- **Data Exchange**: Una alternativa diferente es aquella de transformar los datos en la base de datos original (y que por lo tanto se encuentran organizados de acuerdo al esquema de la misma) de forma tal que se amolden a un nuevo esquema (el esquema objetivo) que es al que nos interesa acceder, pero donde la nueva información (aquella bajo el esquema objetivo) es una representación fiel de la original. El problema no es trivial, incluso cuando disponemos de los mapeos entre esquemas, ya que pueden darse los casos donde no podemos encontrar una instancia de tales mapeos dadas las restricciones impuestos, o bien que existan muchas (quizás infinitas) de tales instancias, donde se impone elegir la mejor de acuerdo a nuestras necesidades.

La diferencia entre los enfoques puede verse en la Figura 7.3.

7.1.2. Aspectos de representación de conocimiento en bases de datos relacionales: expresando bases de datos relacionales como ontologías Datalog$^{\pm}$

Como hemos explicado, en el objetivo a largo plazo del plan de investigación en el cual se enmarca la presente tesis se propone el uso de bases de datos relacionales como repositorios de datos, donde el acceso a los mismos se realiza a través de una federación.

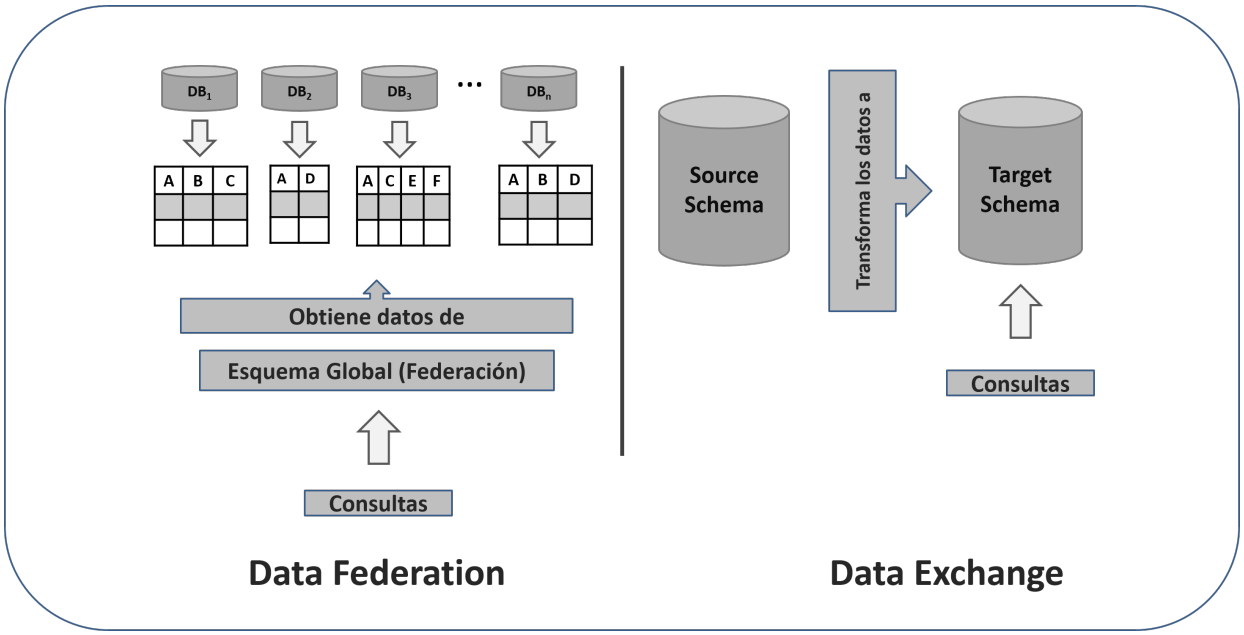

Figura 7.3: Distintos enfoques para la combinación de bases de datos relacionales

Sin embargo, los operadores propuestos en esta disertación trabajan sobre ontologías Datalog$^{\pm}$, por lo que su aplicación directa en la obtención de la federación no es posible. Es por esto que, como primer paso en la consecución de la integración de las bases de datos relacionales a través del uso de operadores de consolidación de ontologías Datalog$^{\pm}$ se propone expresar las BDs originales a través de ontologías Datalog$^{\pm}$.

Las bases de datos relacionales sobre las que trabajaremos estarán entonces representadas a través de los dos componentes de una ontología Datalog$^{\pm}$. A continuación ahondamos en cómo puede ser lograda una representación adecuada para nuestros fines de detección y corrección de conflictos donde, por ejemplo, no necesitamos transformar los datos. Claramente, tal representación dependerá de la base de datos particular que estemos queriendo representar; como fue explicado, tanto en la federación como en el intercambio de datos un paso crucial es la definición de mapeos, involucrando a ingenieros de conocimiento en el proceso. El objetivo de esta parte no es, por lo tanto, el mostrar una representación particular, sino mostrar ejemplos de las herramientas de las que tal ingeniero dispondría al momento de representar una base de datos relacional a través de una ontología Datalog$^{\pm}$. Para ejemplificar cómo tales transformaciones pueden ser llevadas a cabo utilizaremos una base de datos relacional ampliamente conocida, la base de datos Northwind [Nor]. El esquema de tal base de datos es mostrado en la Figura 7.4.

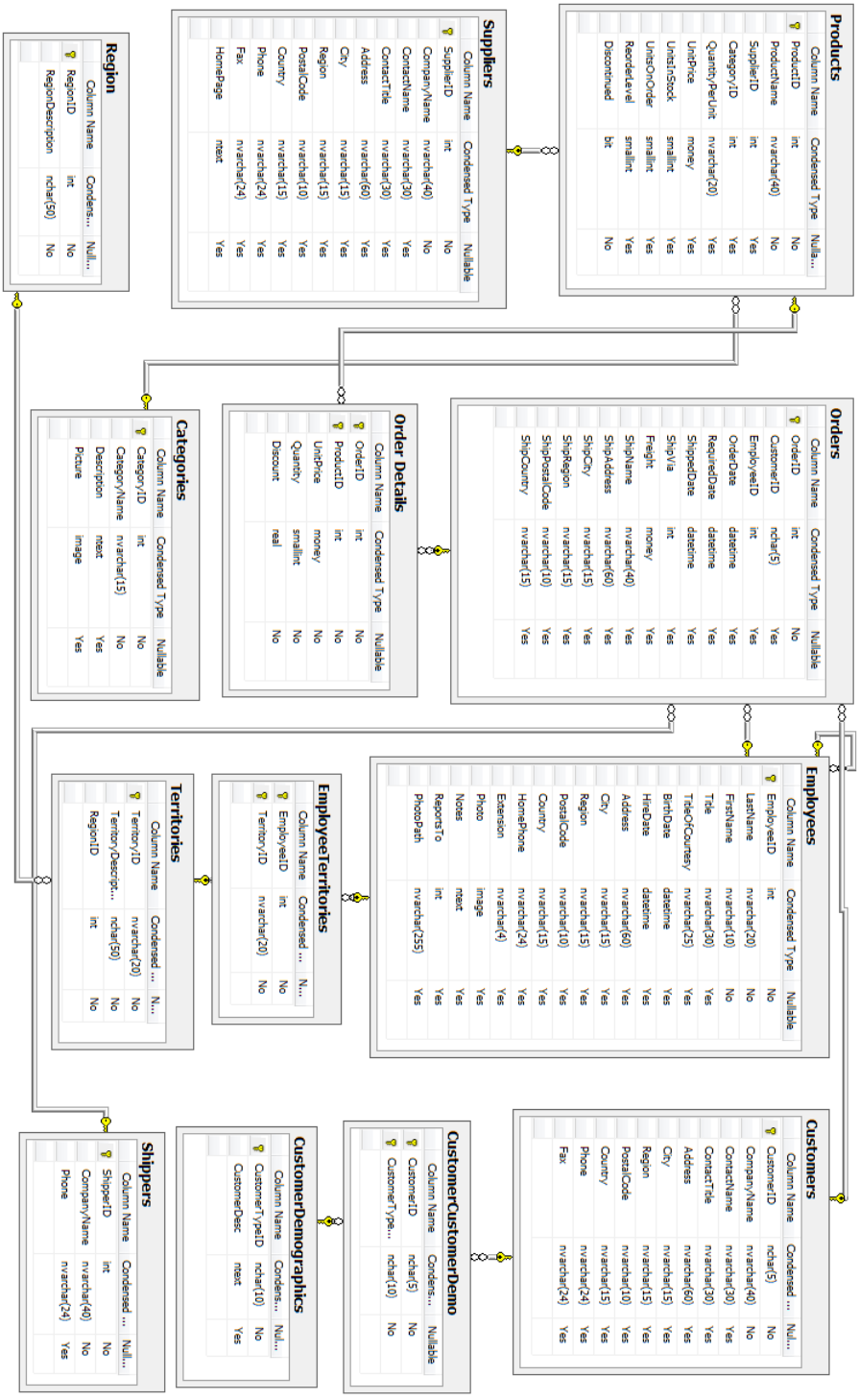

Figura 7.4: Esquema de la base de datos Northwind

Instancia de la base de datos

La representación de las tuplas en la base de datos no representa mayores problemas: dado que tanto las tuplas en las bases de datos relacionales como los átomos en el componente D de una ontología Datalog$^{\pm}$ son instancias de un esquema relacional, la representación es bastante directa. En particular, utilizaremos el nombre de la tabla como el nombre de predicado del átomo, mientras que los distintos valores en los campos serán los términos del átomo.

Ejemplo 7.1 *Considere la tabla Shippers en Figura 7.4. Para representar la información almacenada en tal tabla podríamos tener el átomo* $shippers(ShipperID, CompanyName, Phone)$. *De esta forma, tendríamos el átomo* $shippers(1, \text{“Mediterranean Shipping Company”}, \text{“+54 11 5300 7200”})$ *para identificar la tupla de la tabla Shippers que hace referencia a la compañía con ID = 1.*

Restricciones de integridad

Otro componente que el ingeniero de conocimiento necesita expresar son las restricciones que son impuestas a los datos en la base de datos.

Las distintas reglas acerca de la integridad de la base de datos y validación de datos en ella; como ser integridad referencial, restricciones definidas por el usuario o dependencias funcionales, pueden ser expresadas como fórmulas lógicas a través del componente Σ de la ontología Datalog$^{\pm}$. Claramente, diferentes tipos de restricciones imponen el uso de diferentes tipos de reglas ontológicas de Datalog$^{\pm}$, ya sean TGDs, EGDs o NCs (o quizás combinaciones de las mismas).

Una aclaración importante al respecto del uso de restricciones en bases de datos relacionales tiene que ver con la semántica de las mismas en relación con la suposición de mundo cerrado tradicionalmente usada en este área. Al utilizar tal suposición la semántica de las restricciones de integridad es la de chequeos que deben ser satisfechos por la información explícitamente presente en instancia de base de datos. Esto se contrapone con el modelado de restricciones de integridad mediante reglas en Datalog$^{\pm}$, ya que el conocimiento en tal lenguaje es concebido con una suposición de mundo abierto, tal como suele suceder en los lenguajes ontológicos. De todas formas, ciertos chequeos pueden realizarse en Datalog$^{\pm}$ mediante el uso de EGDs y NCs, y las restricciones y su semántica

pueden ser en mayor o menor medida preservadas para que sirvan a los fines de combinar el conocimiento en pos de obtener la vista única. De hecho, si bien en la presente tesis elegimos no restringir la semántica de las reglas en Datalog$^{\pm}$, han surgido recientemente nuevos enfoques que permiten la utilización de conocimiento ontológico que responde a la suposición de mundo abierto para el chequeo de restricciones y el reconocimiento de mundo cerrado [Pat14, Pat15, PUSC12], que acercaría aún más las bases de datos y las ontologías, permitiendo una representación más directa. A continuación proponemos como usar los componentes de Datalog$^{\pm}$ para representar restricciones.

- Check Constraints: Una restricción de chequeo es un tipo de restricción que especifica un requerimiento que debe ser cumplido por cada tupla en una tabla. Para representarlas deberíamos de utilizar diferentes estrategias, dependiendo del tipo de chequeo necesario. Por ejemplo, a través del uso de Negative Constraints, *v.g.,* para establecer que la columna UnitPrice de la tabla OrderDetails debe almacenar un valor mayor que 0 podríamos definir la NC

  $$orderDetails(OrderID, ProductID, UnitPrice, Quantity, Discount) \wedge UnitPrice \leq 0 \rightarrow \bot$$

  Claramente, el predicado $UnitPrice \leq 0$ puede ser reformulado como $leq(UnitPrice, 0)$ en caso de ser necesario[1].

- Foreign Key Constraints: La integridad referencial es una propiedad de los datos la cual, cuando es satisfecha, requiere que cada valor de un atributo (columna) en una relación (tabla) exista como valor del atributo en una relación (tabla) diferente.

  Para que la integridad referencial sea respetada en una base de datos relacional, todo campo en una tabla que sea declarado como una clave foránea (foreign key - FK) puede contener o bien un valor nulo o sólo valores de la clave primaria o candidata de una tabla padre. En otras palabras, cuando el valor de una clave foránea es usado este debe referenciar una clave primaria válida en la otra tabla [CMR12].

[1]Predicados como *leq* son tradicionalmente asumidos en la literatura como built-in, operaciones predefinidas proporcionadas por el sistema de apoyo.

Para representar en Datalog$^{\pm}$ tales restricciones es posible utilizar Tuple-Generating Dependencies, *v.g.,* considerando la clave foránea ProductID en la tabla OrderDetails, que referencia la clave primaria ProductID en la tabla Products[2].

orderDetails(OrderID, ProductID, ...) $\rightarrow$
$\exists$ProductName products(ProductID, ProductName,...)

Éste es un ejemplo donde podemos notar la influencia de la diferencia entre las suposiciones de mundo cerrado y abierto. En la semántica tradicional de bases de datos bajo la suposición de mundo cerrado una violación a tal restricción se produciría si no existiera el producto en cuestión que es referenciado en la tupla de la tabla OrderDetails. Utilizando la suposición de mundo abierto, en cambio, la restricción permite un completamiento futuro correcto de la base de datos gracias a las reglas con existenciales en la cabeza.

- Not Null Constraints: son restricciones de chequeo particulares que se aseguran que un campo de una tabla tenga un valor conocido, como oposición al valor nulo. Estas restricciones pueden modelarse también a través de NCs, por ejemplo, el hecho de que el CompanyName de la tabla Shippers no pueda ser vacío podemos expresarlo como

$$\text{shippers(ShipperID, CompanyName, Phone)} \wedge \text{ CompanyName} = \text{“null”} \rightarrow \bot$$

- Unique Constraints: estas restricciones indican que el valor en una columna es único, es decir, que no hay dos tuplas con el mismo valor para esa columna. Claramente, estas restricciones tienen una correspondencia directa con las Equality-Generating Dependencies en Datalog$^{\pm}$, por lo cual es posible utilizar éstas últimas para representar las primeras. Por ejemplo, el campo ShipperID de la tabla Shipper es un campo único, lo que representamos mediante la EGD

shippers(ShipperID, CN, P)$\wedge$shippers(ShipperID', CN', P') $\rightarrow$ ShipperID = ShipperID'

[2]Por razones de legibilidad sólo nombramos algunos de aquellos campos involucrados, pero claramente el resto también estaría. Por ejemplo, si tenemos products(ProductID, ProductName, ...) debe entenderse en realidad un átomo de nombre products con todos los demás campos de la tabla Products, además de los nombrados (incluyendo la cuantificación existencial para los campos que corresponda).

Por razones de simplicidad los acortamos, pero CN y P corresponden con los campos CompanyName y Phone respectivamente.

Diferente es el caso donde aquello que no puede repetirse es la combinación de dos o más campos, como es el caso de las claves primarias en tablas relacionales. Por ejemplo, consideremos la tabla EmployeeTerritories, que relaciona a un empleado con el territorio que tiene asignado. Para tal tabla, aquello que no puede repetirse es la combinación de ambos campos en la tabla, *esto es,* PK = EmployeeID + TerritoryID (pero, por ejemplo, un empleado puede tener más de un territorio asignado, y por lo tanto su ID aparecer en más de una tupla). Para estos casos, previamente al uso de la EGD para el chequeo de unicidad necesitamos identificar de alguna manera que la concatenación de ambos campos es lo que no puede repetirse. Para ello, una posibilidad es la de crear mediante una TGD un átomo auxiliar que represente al par, y luego aplicar la restricción sobre tal átomo a través de la EGD. Es decir, esto podría ser representado a través de las siguientes fórmulas:

$$\text{employeeTerritories(EID, TID)} \rightarrow \exists PK\ \text{empTerrPK(EID, TID, PK)}$$

$$\text{empTerrPK(EID, TID, PK)} \wedge \text{empTerrPK(EID, TID, PK')} \rightarrow \text{PK} = \text{PK'}$$

- Restricciones de entidad: el modelo relacional establece que toda relación (o tabla) debe poseer un identificador, llamado la clave primaria (primary key - PK), de forma tal que cada tupla de una misma relación pueda ser identificada por su contenido, esto es, por un valor único y mínimo. De esta forma, como las Primary Key Constraints son campos Unique y Not Null, podemos utilizar el mismo modelado que para estas restricciones. Es decir, por cada PK tendremos (al menos) una EGD y una NC, para el chequeo de unicidad y el de nulidad, respectivamente.

- Dependencias Funcionales: en la teoría de bases de datos relacionales, una dependencia funcional (functional dependency - FD) es una restricción entre dos conjuntos de atributos (campos) en una relación de la base de datos. Dada una relación R, se dice que un conjunto de atributos X determina funcionalmente otro conjunto de atributos Y (denotado $X \rightarrow Y$) si, y sólo si, cada valor de X es asociado con precisamente un valor de Y [HM10, Dat12]. En términos más sencillos, si los valores de X son conocidos (por ejemplo, supongamos que son x), entonces los valores para los

atributos Y correspondientes pueden ser determinados mirando en aquellas tuplas en la relación que contienen a x. De esta manera, una dependencia $X \rightarrow Y$ significa que los valores de Y son determinados por los valores de X.

Con respecto a cómo las dependencias funcionales pueden ser expresadas en Datalog$^{\pm}$, en la presente tesis proponemos que ésto se haga utilizando TGDs.

Por ejemplo, asumamos a fines de ejemplificar lo expresado que dada una orden de compra (es decir, una tupla en la tabla OrderDetails) el descuento que corresponde a tal compra (campo Discount) se determina de acuerdo al precio unitario (campo UnitPrice) y a la cantidad ordenada (campo Quantity). Es decir, tenemos la FD OrderDetails.UnitPrice $\wedge$ OrderDetails.Quantity $\rightarrow$ OrderDetails.Discount.

Para representar esta situación podemos usar la siguiente TGD.

orderDetails.UnitPrice(UP) $\wedge$ orderDetails.Quantity(Q) $\rightarrow$
$\exists$D orderDetails.Discount(D)

Resumen - Representación de bases de datos relacionales a través de ontologías Datalog$^{\pm}$

En resumen, una posibilidad para el modelado de bases de datos relacionales a través de ontologías Datalog$^{\pm}$ es la mostrada en la siguiente tabla.

Bases de Datos	**Ontologías Datalog$^{\pm}$**
Instancia del esquema	Componente D
Dependencias Funcionales	TGDs
Check Constraints	NCs o EGDs
FK Constraints	TGDs
Not Null Constraints	NCs
Unique Constraints	(TGDs +) EGDs

7.1.3. Usando la consolidación de ontologías Datalog$^{\pm}$ como medio de combinación de bases de datos relacionales en la obtención de la vista única

Hemos mostrado como pueden utilizarse ontologías Datalog$^{\pm}$ para representar bases de datos relacionales. A continuación, procedemos a mostrar como continúa el proceso de

la creación de la vista única (federación); esto es, como los operadores de consolidación son utilizados como soporte en tal creación.

Para lograr tal cometido, una vez que todas las bases de datos relacionales involucradas han sido representadas a través de las correspondientes ontologías Datalog$^{\pm}$, procedemos a unir las mismas mediante unión conjuntista (Figura 7.2), obteniendo de esta forma una única ontología con toda la información proveniente de las bases de datos originales. Claramente, tal ontología es propensa a tener conflictos tanto de consistencia como de coherencia, especialmente al integrar información de diferentes fuentes. Es allí donde entran en juego los operadores definidos en la presente tesis: la identificación y eliminación de tales conflictos.

Esto es, supongamos que tenemos un conjunto $\Phi = \{DB_1, DB_2, \ldots, DB_n\}$ de bases de datos relacionales (con suposición de nombre único y un lenguaje común entre las mismas), y sea Δ un operador tal que $\Psi = \{KB_1, KB_2, \ldots, KB_n\} = \Delta(\Phi)$ es el conjunto de ontologías Datalog$^{\pm}$ obtenidas mediante una transformación como la mostrada en la Sección 7.1.2. Entonces, dado el conjunto de ontologías Ψ tenemos que la obtención de la vista única de las bases de datos en Φ mediante la aplicación de un operador $\Upsilon_{\rho,\varrho}$ de consolidación de ontologías Datalog$^{\pm}$ (ya sea definido como en la Definición 5.6 o en la Definición 6.6) es igual a la Ecuación 7.1.

$$\Theta(\Phi) = \Upsilon(\bigcup_{KB_i \in \Psi} (KB_i)) \tag{7.1}$$

Es sencillo ver que el operador definido como $\Upsilon(\bigcup_{KB_i \in \Psi} (KB_i))$ es en realidad un operador de combinación de ontologías Datalog$^{\pm}$, similar a la clase de operadores de combinación de creencias presentada en la Sección 4.4.2, y en particular aprovecha la interdefinibilidad en los operadores presentada en la Definición 4.8. Esto es, el operador toma un conjunto de ontologías Datalog$^{\pm}$, y devuelve una única ontología, eliminando para ello cada conflicto que encuentre en la unión de tales ontologías. A su vez, claramente, la ontología $\Theta(\Phi)$ obtenida finalmente posee todas las características propias de las ontologías obtenidas a través de los operadores de consolidación definidos en la presente tesis. En particular, la ontología obtenida es coherente y consistente, resolviendo todos aquellos conflictos que pudieran surgir de la integración de información proveniente de diversas fuentes. De esta manera, obtenemos una vista única libre de conflictos que puede ser accedida como una federación de las bases de datos originales.

7.1.4. Refinando la obtención de la vista: definición de operadores de integración a partir de los operadores de consolidación

Hemos visto recientemente como es posible definir operadores de combinación de ontologías Datalog$^{\pm}$ a través del uso de consolidación, y como tales operadores pueden ser utilizados en la obtención de una vista única de las ontologías en juego (y por ende de las bases de datos relacionales que las mismas representan). Sin embargo, como hemos mencionado en el Capítulo 4 tales operadores, al operar sobre la unión de las bases de conocimiento originales, no pueden utilizar información acerca de las fuentes de la información a la hora de resolver conflictos [Kon00]. Distinto es el caso de los operadores de integración, donde la información acerca del origen de las distintas piezas de información es preservado (o al menos utilizado en la resolución de conflictos). Claramente, dependiendo del entorno de aplicación el conocer la fuente original de la información puede reportar beneficios o no. En este sentido, para nuestro fin (esto es, la obtención de una vista única) tal información puede ser utilizada de manera que refinemos el proceso recientemente presentado. A continuación mostraremos como operadores de integración pueden ser obtenidos mediante nuestros operadores de consolidación. La estrategia básica se basa en utilizar instancias particulares de la relación $\preccurlyeq$, adaptadas a las necesidades particulares de este caso.

Estados epistémicos aumentados

Como hemos explicado en el Capítulo 5, los operadores de consolidación aquí presentados tienen como estado epistémico a *una* ontología Datalog$^{\pm}$. Esto mismo sucede con los operadores de combinación presentados recientemente, ya que utilizan la unión de las ontologías originales. Claramente, el poseer solamente una ontología como estado epistémico hace imposible la consecución de operadores de integración, ya que no podemos mantener información acerca de las diferentes fuentes de información.

Para paliar tal situación utilizaremos una extensión a tales estados epistémicos para incluir información pertinente de las fuentes originales. Esto es, en lugar de que el estado epistémico sea simplemente una ontología Datalog$^{\pm}$ KB, para nuestros operadores de integración asumiremos un *estado epistémico aumentado*, que son pares $(KB, \circledast(\Psi))$, donde el componente $\circledast(\Psi)$ es una matriz de información acerca de los átomos y las TGDs

en KB calculada mirando en el conjunto de ontologías originales Ψ. Tal información es obtenida, entonces, a través de una función $\circledast$ que toma como parámetro el conjunto de ontologías a integrar y devuelve la matriz llena con la información pertinente. Obviamente, diferentes funciones $\circledast$ pueden ser elegidas para adecuarse a distintas necesidades. Por ejemplo, podríamos tener una función que nos devuelve un recuento de las veces que determinado átomo aparece en las distintas fuentes originales (generando de esta manera un vector de tamaño igual a la cantidad de fórmulas donde cada fórmula guarda en su posición correspondiente tal recuento). Otra opción sería definir una función que para cada fórmula nos devuelva la credibilidad asociada a la fuente de información de la misma; de esta forma, como resultado tendríamos un vector nuevamente de tamaño n (donde n es igual a la cantidad de fórmulas en KB) pero que esta vez almacene en la posición correspondiente el máximo (o mínimo, si es nuestra preferencia) entre los valores de credibilidad de las ontologías que efectivamente poseen tal fórmula. Incluso, podríamos tener una función que combine las dos anteriores, devolviendo una matriz de $2n$ donde cada dimensión corresponda ya sea con el recuento o la credibilidad de la fórmula.

Utilización de la información adicional para determinar el comportamiento de la consolidación

Hasta ahora hemos visto cómo aumentar los estados epistémicos de los operadores de consolidación para retener información acerca de las fuentes de origen de las distintas piezas de información en la ontología que es la unión de éstas. A continuación analizamos cómo tal información puede ser aprovechada para refinar el comportamiento de los operadores de consolidación, obteniendo de esta manera operadores de integración de ontologías Datalog$^{\pm}$ que puedan respetar los requerimientos tradicionales de tales operadores [KP02].

Cómo es de esperarse, la clave para lograr tal objetivo está en manipular la relación $\preccurlyeq$ para que la misma se adapte a las necesidades particulares del entorno de aplicación de forma que los principios de integración sean respetados. Por ejemplo, uno de los tipos de operadores de integración más importantes es el de *mayoría* (majority) [KP02] el cual establece, como su nombre lo indica, que llegado el caso de conflicto entre dos o más fórmulas deben retenerse aquellas que mayor consenso tengan (esto es, las que aparezcan más veces) entre las fuentes originales siendo integradas. En lo que resta de la Sección 7.1.4 nos centraremos, a modo de ejemplo, en cómo caracterizar operadores de integración que se guíen por el principio de mayoría mientras que utilizan para la base de su defini-

ción los operadores de consolidación introducidos en el presente trabajo, pero operadores alternativos pueden definirse de manera sencilla. Para definir operadores de integración por mayoría basta con definir a la relación $\preccurlyeq$ de la siguiente manera.

Definición 7.1 (Relación de Preferencia entre fórmulas por Mayoría) *Sea $\mathcal{L}$ un lenguaje de primer orden, $\Psi = \{KB_1, KB_2, \ldots, KB_n\}$ un conjunto de ontologías Datalog$^{\pm}$ sobre $\mathcal{L}$, $\alpha, \beta \in \bigcup_{KB_i \in \Psi} (KB_i)$ dos fórmulas en Ψ y $\circledast$ una función tal que $\circledast(\alpha)$ es el número de ontologías en Ψ que contienen a α. Una relación de preferencia $\preccurlyeq_{\mathfrak{m}}$ entre las fórmulas en $\mathcal{L}$ basada en mayoría es una relación total, reflexiva, antisimétrica y transitiva en $\mathcal{L}$ tal que se verifica que $\alpha \preccurlyeq_{\mathfrak{m}} \beta$ sí y solo sí $\circledast(\alpha) < \circledast(\beta)$.*

La particularización propuesta en la Definición 7.1 es sólo un ejemplo, y como hemos mencionado previamente otras funciones $\circledast$ pueden usarse de manera similar para definir órdenes alternativos.

Una vez que tenemos definido el orden particular que utilizaremos ya nos encontramos en condiciones de definir nuestros operadores de integración basados en consolidación, debido a la modularidad del uso de $\preccurlyeq$ en la definición de las funciones de incisión. Por lo tanto, tenemos que un operador de integración por (en este caso) mayoría es definido como aquél operador de consolidación que opera sobre la unión de las ontologías originales utilizando el orden particular introducido en la Definición 7.1 para la definición de las funciones de incisión a utilizar.

Nuevamente, sea $\Phi = \{DB_1, DB_2, \ldots, DB_n\}$ de bases de datos relacionales, y sea Δ un operador tal que $\Psi = \{KB_1, KB_2, \ldots, KB_n\} = \Delta(\Phi)$ es el conjunto de ontologías Datalog$^{\pm}$ obtenidas mediante una transformación como la mostrada en la Sección 7.1.2. Sea $\preccurlyeq_{\mathfrak{m}}$ una relación de preferencia entre fórmulas por Mayoría, y $\rho_{\preccurlyeq_{\mathfrak{m}}}$ una función de incisión en restricciones y $\varrho_{\preccurlyeq_{\mathfrak{m}}}$ una función de incisión en datos que usan $\preccurlyeq_{\mathfrak{m}}$, respectivamente. Finalmente, sea $\Upsilon_{\rho_{\preccurlyeq_{\mathfrak{m}}}, \varrho_{\preccurlyeq_{\mathfrak{m}}}}$ un operador de consolidación definido en base a $\rho_{\preccurlyeq_{\mathfrak{m}}}$ y $\varrho_{\preccurlyeq_{\mathfrak{m}}}$. El operador de integración por mayoría basado en consolidación es definido como en la Ecuación 7.2.

$$\Theta_{\mathfrak{m}}(\Phi) = \Upsilon_{\rho_{\preccurlyeq_{\mathfrak{m}}}, \varrho_{\preccurlyeq_{\mathfrak{m}}}}(\bigcup_{KB_i \in \Psi} (KB_i)) \tag{7.2}$$

Claramente, si bien el operador $\Upsilon_{\rho_{\preccurlyeq_{\mathfrak{m}}}, \varrho_{\preccurlyeq_{\mathfrak{m}}}}$ opera directamente sobre la unión de ontologías, el mismo posee a través de $\preccurlyeq_{\mathfrak{m}}$ la información relevante respecto de las fuentes originales

de las distintas piezas de información (para el caso presentado la cantidad de ontologías conteniendo la fórmula), y por lo tanto el mismo se adapta a la intuición de los operadores de integración. Tal tipo de operador puede entonces ser utilizado en la manera presentada en la Sección 7.1.3 para la obtención de una vista única donde la misma sea más refinada, de acuerdo a los aspectos que codifiquemos a través de la relación $\preccurlyeq_{\mathtt{m}}$.

7.2. Uso de la consolidación como soporte a procesos de evolución de ontologías

La evolución de ontologías (ontology evolution) es un proceso que versa acerca de la dinámica de conocimiento en las ontologías. Básicamente, se trata de modificar las ontologías para reflejar cambios en el dominio que las mismas representan de forma tal que no haya conflictos en la ontología resultante. En la presente sección propondremos un escenario donde los operadores de consolidación definidos en esta disertación pueden apoyar la definición de procesos de evolución de ontologías. Para lograr esto, primeramente veremos cómo los operadores de consolidación pueden ser la base de operadores de revisión de ontologías Datalog$^{\pm}$ en general, y en particular operadores de revisión priorizados.

Luego, haremos una breve introducción a la evolución de ontologías; y finalmente nos ocuparemos de mostrar cómo mediante tales operadores de revisión puede lograrse uno de los objetivos más importantes en cualquier proceso de evolución de ontologías: la consecución de la evolución de forma libre de conflictos.

7.2.1. Operadores de revisión en Datalog$^{\pm}$

En la presente tesis hemos presentado operadores de consolidación que se basan en la contracción de las ontologías de una fórmula particular, la falacia $\bot$. Generalizar esta contracción a un átomo cualquiera no representa mayores problemas; para ver esto más claramente considere el siguiente ejemplo.

Ejemplo 7.2 *Dada la ontología*

$$KB = \left\{ \begin{array}{rl} D\text{:} & \{a_1 : en\_terapia(charlie), a_2 : en\_relacion(kate, charlie), \\ & a_3 : paciente(charlie)\} \\ \Sigma_{NC} : & \{\tau_1 : tratando(T, P) \wedge en\_relacion(T, P) \rightarrow \bot\} \\ \Sigma_{E} : & \{\nu_1 : tratando(T, P) \wedge tratando(T', P) \rightarrow T = T'\} \\ \Sigma_{T} : & \{\sigma_1 : paciente(P) \rightarrow en\_terapia(P)\} \end{array} \right\}$$

Podemos realizar la contracción por el átomo $en\_terapia(charlie)$ simplemente removiendo al mismo, y a aquellos otros átomos que puedan ser usados para derivarlo, de forma tal que la KB resultante no lo infiera. En este caso podríamos tener por ejemplo

$$KB \div a_1 = KB \setminus \{a_1, a_3\}$$

donde $\div$ es el operador de contracción que remueve de los a_1-kernels (conjuntos mínimos que derivan a_1) alguna fórmula.

Sin embargo, si queremos plantear un operador de revisión en base a uno de contracción como el presentado y la identidad de Levi presentada en el Capítulo 4 no podemos, ya que tendríamos

$$KB * a_1 = (KB \div \neg a_1) + a_1$$

Claramente, la expansión no es problema, simplemente se agrega el átomo a_1 a la KB resultante de la contracción. Sin embargo, justamente es la contracción la que nos plantea un problema: en Datalog$^{\pm}$ no existe la negación, por lo que no es claro qué significaría $\neg a_1$ en este contexto (no existe en Datalog$^{\pm}$ algo como $\neg en\_terapia(charlie)$). El problema es entonces cómo definir operadores de revisión en Datalog$^{\pm}$.

El hecho de que exista una forma de contracción nos muestra que quizás es posible realizar la revisión en términos de contracción, incluso en la situación de que no tengamos negación. Se podría pensar en realizar revisión externa [Han97b, Han01], y definir la revisión como

$$KB * a_1 = (KB + a_1) \div \neg a_1$$

Pero volvemos a tener el problema de la negación de a_1. Sin embargo, el hecho de que los operadores de consolidación propuestos en la presente tesis tomen como estado epistémico bases de creencias en lugar de conjuntos de creencias nos da la posibilidad de manejar estados intermedios inconsistentes. Por lo tanto, en base a estos operadores que utilizan bases de creencias en Datalog$^{\pm}$ es posible definir un operador de revisión externa por una fórmula α como la secuencia:

1. expandir por α
2. contraer por $\bot$

Por lo tanto, es nuestra propuesta el utilizar el enfoque de consolidación presentado en esta disertación como la base del desarrollo de operadores de revisión de ontologías Datalog$^{\pm}$. Para esto, tales operadores comienzan agregando la fórmula en consideración (ya sea un átomo o una TGD) para luego asegurarse de que lo resultante de tal expansión no sea inconsistente/incoherente mediante la remoción de átomos o TGDs. Claramente, la consolidación podría ser realizada por operadores como los definidos en las Definiciones 5.6 y 6.6. Es decir, tendríamos

$$KB * a_1 = (KB + a_1) \div \bot = (KB + a_1)!$$

donde el operador de consolidación ! corresponde con alguno de los enfoques propuestos en este trabajo.

De esta forma podríamos definir operadores de revisión con una semántica que es de alguna manera análoga a la de los operadores tradicionales (no priorizados, como los operadores de Semi-Revision presentados por Hansson en [Han97b]) para este entorno sin negación, ya que si existe un subconjunto A de KB tal que a_1 esta en conflicto con A (por ejemplo a través de una NC) entonces podríamos considerar que A es equivalente a $\neg a_1$ (ya que juntos derivan $\bot$). De esta forma, como o bien a_1 o A pertenecen a la revisión pero no ambos, entonces tales operadores se comportan de forma similar a operadores de revisión no priorizados [Han97b].

Sin embargo, es evidente que a los fines de su utilización como la base de la definición de procesos de evolución de ontologías, los operadores de revisión no priorizados no resultan del todo adecuados: como los mismos pueden escoger rechazar la entrada epistémica, entonces puede darse el caso de que aquella fórmula que estamos queriendo introducir en la ontología no termine siendo parte de la misma. Evidentemente, esto violaría un principio fundamental de la evolución de ontologías: al querer reflejar un cambio ya ocurrido en el mundo, en la evolución de ontologías se requiere que aquella formula que disparó el proceso de cambio sea parte de la ontología resultante. Es decir, la entrada epistémica se asume verdadera y se requiere que sea aceptada (similar a lo que ocurre en el enfoque AGM). De todas maneras, la transformación de operadores no priorizados a priorizados no representa gran dificultad en el framework propuesto en la presente tesis. Nótese que los operadores son generales, y sólo responden a una relación $\preccurlyeq$ que no se encuentra instanciada. Por lo tanto, adecuando la relación $\preccurlyeq$ usada por los operadores para que siempre escoja retener a la entrada epistémica (en el caso que hemos discutido previamente, que se dé que $A \preccurlyeq a_1$) podemos incluso definir operadores de revisión priorizados análogos a operadores clásicos como los de Partial Meet Revision y similares, los cuales serán más adecuados para su utilización en procesos de evolución de ontologías Datalog$^{\pm}$.

7.2.2. Uso de los operadores de revisión para el soporte a la evolución de ontologías Datalog$^{\pm}$

Hemos visto como los operadores de consolidación definidos en la presente tesis pueden servir de base en la definición de procesos de revisión de ontologías Datalog$^{\pm}$. A continuación veremos como tales operadores pueden ser utilizados en la definición de métodos de *evolución de ontologías*.

A grandes rasgos podemos definir a la evolución de ontologías (adaptando una definición dada por Stojanovic *et al.* [SMMS02, Sto04]) como "la adaptación en tiempo y forma de una ontología a los cambios que surjan en el entorno y la propagación *consistente* de tales cambios". Es decir, la evolución de ontología no consiste simplemente en agregar un nuevo axioma o átomo a una ontología, sino que tal acción debe ser acompañada de las modificaciones necesarias para se mantengan las propiedades deseables de la ontología una vez que el proceso finaliza.

La importancia de la evolución de ontologías radica en que, tradicionalmente, en el campo de las ontologías la gran mayoría de los trabajos se han enfocado en problemas en la etapa de construcción y desarrollo de las ontologías, por lo general asumiendo que el conocimiento del dominio encapsulado a través de la ontología no cambia con el tiempo. Sin embargo, en un entorno de aplicación abierto y dinámico, el conocimiento del dominio evoluciona constantemente [Fen01]. Por lo tanto, el desarrollo de ontologías es un proceso dinámico el cual comienza con una ontología inicial (posiblemente incompleta, un prototipo de la ontología final buscada), la cual es subsecuentemente revisada, refinada y modificada para incluir cada vez más detalle [NM$^+$01].

Es decir, la evolución de ontologías, como su nombre lo indica, se refiere al *dinamismo* del conocimiento. Es fácil de comprender entonces la estrecha relación que ésta mantiene con la Teoría de Cambios y la revisión de creencias. Se podría argumentar que, a cierto nivel, *la evolución de ontologías es al campo de las ontologías lo que Teoría de Cambios es al campo de Representación de Conocimiento*. Esto es, tanto la evolución de ontologías como la revisión de creencias se encargan de ver cómo el paso del tiempo puede afectar a cosas (conocimiento) que suelen ser considerados estáticos por los otros enfoques en sus respectivas áreas. De todas formas creemos oportuno aclarar que no es el espíritu de esta tesis el afirmar que ambos campos son equivalentes: evidentemente, si bien comparten el espíritu de captura de dinamismo, también poseen diferencias que hacen que los mismos se adapten a sus respectivas áreas de manera diferente. Sin embargo, el hecho de que compartan ciertos objetivos hace que, como veremos a continuación, los operadores de revisión de ontologías Datalog$^\pm$ (definidos a través de consolidación como fue explicado) puedan ser utilizados para capturar cambios en el conocimiento modelado por la ontología.

En su trabajo, Stojanovic *et al.* [SMMS02, Sto04] formulan una serie de requerimientos de diseño que debe respetar todo proceso adecuado de evolución:

(a) debe permitir realizar los cambios necesarios a la ontología y asegurarse de la consistencia de la misma a pesar de tales modificaciones;

(b) debería ser un proceso supervisado que permita administrar los cambios de manera sencilla;

(c) y debe facilitar la continuidad del refinamiento de la ontología (iteración de evolución).

Claramente, el primer punto dentro de los requerimientos de Stojanovic *et al.* guarda una estrecha relación con los operadores de revisión de ontologías Datalog$^{\pm}$. Esto es, supongamos que tenemos un operador $*$ de revisión de ontologías Datalog$^{\pm}$ como los descriptos anteriormente, en su versión priorizada para asegurar que la información más actual es siempre retenida en caso de conflicto. Claramente, si somos informados de un cambio en el conocimiento podríamos usar tal operador para introducir de manera segura tal cambio. Para ver tal situación consideremos el siguiente ejemplo.

Ejemplo 7.3 *Considere la siguiente ontología Datalog$^{\pm}$.*

$$KB = \left\{ \begin{array}{rl} D\text{:} & \{a_1 : tiempo(cordoba, llueve), a_2 : tiempo(jujuy, soleado), \\ & a_3 : seco(jujuy), a_4 : humedo(parana)\} \\ \Sigma_{NC}: & \{\tau_1 : tiempo(L, llueve) \wedge calor(L) \rightarrow \bot\} \\ \Sigma_{E}: & \{\nu_1 : tiempo(L, T) \wedge tiempo(L, P') \rightarrow P = P'\} \\ \Sigma_{T}: & \{\sigma_1 : seco(L) \rightarrow calor(L)\} \end{array} \right\}$$

Supongamos que el nuevo reporte meteorológico nos informa que ha comenzado a llover en Jujuy. Por lo tanto, deberíamos revisar nuestra ontología para reflejar este acontecimiento, es decir, deberíamos realizar la operación

$$KB * tiempo(jujuy, llueve).$$

Podemos ver que la inclusión directa del átomo $a_5 : tiempo(jujuy, llueve)$ sin llevar a cabo otras acciones adicionales no respeta el primer principio de Stojanovic et al.*, ya que agregar este átomo nos trae aparejado ciertos conflictos, resultando en el hecho de que la ontología deje de ser consistente. Analicemos, sin embargo, cómo se comporta nuestro operador de revisión priorizado basado en consolidación en este caso. Primeramente, como se explico con anterioridad el operador procede a agregar el átomo (creando un estado intermedio de la ontología donde la misma es, en este caso, inconsistente) y chequear si efectivamente la inclusión directa acarrea conflictos. En el caso presentado el operador detecta dos kernels de datos, uno por la violación de τ_1 y el otro por la violación de ν_1. A saber, tenemos*

$$\perp\!\!\!\perp_{(D\cup\{a_5\},KB)} = \{\{tiempo(jujuy, soleado), tiempo(jujuy, llueve)\}, \{seco(jujuy), tiempo(jujuy, llueve)\}\}$$

Siendo el operador de revisión uno priorizado, entonces tenemos que el mismo elimina los átomos a_2 y a_3 para resolver los conflictos, restaurando de esta forma la consistencia, y dando como resultado la siguiente ontología, la cual refleja el cambio en el entorno que nos indica que en Jujuy ha comenzado a llover mientras que se mantiene la consistencia de la ontología, respetando así la máxima de la evolución de ontologías.

$$KB * tiempo(jujuy, llueve) =$$

$$\left\{\begin{array}{rl} D\text{:} & \{a_1 : tiempo(cordoba, llueve), a_4 : humedo(parana), \\ & a_5 : tiempo(jujuy, llueve)\} \\ \Sigma_{NC}: & \{\tau_1 : tiempo(L, llueve) \wedge calor(L) \rightarrow \perp\} \\ \Sigma_{E}: & \{\nu_1 : tiempo(L, T) \wedge tiempo(L, P') \rightarrow P = P'\} \\ \Sigma_{T}: & \{\sigma_1 : seco(L) \rightarrow calor(L)\} \end{array}\right\}$$

Ahora, procedemos a mirar más en profundidad lo que sucede con los requerimientos b y c de Stojanovic *et al.* [SMMS02, Sto04]. Ambos requerimientos pueden ser en cierta medida soportados por un proceso de evolución basado en el uso de operadores de revisión de ontologías Datalog$^{\pm}$.

- Con respecto al punto b, podemos ver que el hecho de manejarnos con conflictos mínimos puede ser una ventaja a la hora de requerir supervisión de cambios por parte de un ingeniero: la herramienta de visualización que soporte el proceso de evolución podría desplegar los distintos kernels al ingeniero, permitiéndole de esta manera visualizar los conflictos que la inclusión de la nueva información trajo aparejados. Es más, como los operadores no están restringidos a una relación $\preccurlyeq$ particular (y además la misma puede ser arbitraria) es posible que dejemos la definición de qué formula eliminar en manos del ingeniero (inclusive la nueva información, haciendo al operador no priorizado si así lo requiriera) y de todas formas la ontología

resultante cumpliría con las propiedades presentadas en los teoremas 5.1 o 6.1, dependiendo del enfoque elegido. Cabe aclarar, sin embargo, que la supervisión puede llegar a ser dificultosa en casos de revisiones que involucren una gran cantidad de conflictos (algo que es probable que ocurra de todos modos sin importar la técnica de supervisión).

- Acerca del punto c, nuevamente el hecho de no atarnos a una relación $\preccurlyeq$ particular puede ayudar a paliar el problema de iteración. Tal problema es ampliamente reconocido por la comunidad de revisión de creencias como una situación difícil de manejar debido a, por ejemplo, que los órdenes que provocan un cambio pueden no mantenerse una vez que el cambio esta realizado. En nuestro caso la iteración del proceso de revisión (y por ende la continuidad de la evolución) es posible. Por ejemplo, nuevamente si consideramos la supervisión por parte de un ingeniero éste mismo podría encargarse, como dijimos recientemente, de las decisiones respecto a la eliminación de formulas. De esta forma, no importa cuantas veces iteremos siempre podremos definir como resolver los nuevos conflictos que vayan apareciendo (y claramente el hecho de usar un supervisor es sólo un ejemplo, otras relaciones de preferencia podrían utilizarse consiguiendo el mismo efecto).

Por todo lo dicho podemos concluir que un proceso de evolución de ontologías Datalog$^{\pm}$ puede ser efectivamente apoyado por medio de los operadores de consolidación de ontologías Datalog$^{\pm}$ presentados en este trabajo, mediante su utilización como la maquinaria que hace surgir operadores de revisión de ontologías Datalog$^{\pm}$.

7.3. Trabajo futuro

A través de este trabajo hemos introducido distintos operadores para la consolidación de ontologías Datalog$^{\pm}$, tanto a nivel de las propiedades esperadas de los mismos como a nivel de posibles construcciones. Durante el desarrollo de estos operadores hemos identificado diferentes líneas alternativas de trabajo, que esperamos seguir en la continuidad del trabajo aquí desarrollado. A continuación presentaremos y explicaremos brevemente las mismas.

7.3.1. Operadores basados en la manipulación de conjuntos maximalmente consistentes

Primeramente, como explicamos previamente en el presente trabajo, para el desarrollo de los operadores aquí presentados nos hemos inspirado y seguido las intuiciones propuestas por Hansson [Han94, Han97b, Han01] las cuales se basan en la manipulación de conjuntos conflictivos mínimos, conocidos como kernels, donde la remoción de alguna fórmula de los mismos resuelve los conflictos. Sin embargo, como hemos explicado en el Capítulo 4, existen otros enfoques posibles, los cuales están fundados principalmente en la teoría AGM [AM85, AGM85].

Al contrario de la propuesta presentada en este trabajo, muchos de esos enfoques atacan los problemas desde el lado opuesto, *esto es,* se basan en la búsqueda y manipulación de subconjuntos maximalmente consistentes de la base de conocimiento. Como tal, una de las líneas de investigación futuras es el desarrollo de nuevos operadores de consolidación de ontologías Datalog$^{\pm}$ los cuales estén basados en los desarrollos AGM, para de esta forma completar el espectro de operadores de consolidación que surjan tanto de la eliminación de conflictos mínimos como de la búsqueda de conjuntos consistentes maximales.

La razón detrás de la búsqueda de completar este espectro es que, como se da cuenta en [Rib13], existe una interesante discusión siendo llevada a cabo acerca de cuál de los enfoques es mejor. En particular, en [Rib13] se muestran ejemplos que muestran que para ciertas instancias es más rápido utilizar contracción basada en la manipulación de kernels, mientras que en otras ocasiones es mejor utilizar el enfoque de manejo de conjuntos maximalmente consistentes propuestos en partial meet contraction. Este hecho de que ambos enfoques puedan ser útiles en diferentes situaciones es la motivación para continuar por esta línea de investigación. Es más, en [Rib13] el autor da cuenta de que la detección de que enfoque es mejor dada una instancia particular es al momento un problema abierto; si tal detección fuera posible entonces contar con ambos tipos de operadores permitiría un mejor desempeño, ya que podríamos mirar la instancia y decidir el enfoque a utilizar, mejorando la performance computacional a la hora de realizar la consolidación.

7.3.2. Modificaciones a la relación de preferencia entre fórmulas

Además del cambio de enfoque explicado en el párrafo anterior, otros posibles operadores que pensamos analizar son aquellos que resulten de utilizar estrategias particu-

lares para determinar que fórmula debe ser removida (al contrario de los presentados en la presente disertación, que utilizan una relación arbitraria); en particular, un enfoque interesante (también proveniente de la tradición AGM) es el de epistemic entrenchment [GM88, Rot92, MLH00].

La intuición detrás de tal enfoque es que, al momento de tomar la decisión de remover una fórmula u otra, aquella que es retenida es la que tiene un mayor nivel de *atrincheramiento* (entrenchment). En general, el entrenchment está basado en algún orden dado para las fórmulas que indica qué tan importantes son las mismas; como trabajo futuro pensamos ir particularizando de distintas formas como tal orden es obtenido (*esto es,* ir particularizando la relación $\preccurlyeq$) para ir definiendo nuevos operadores que se adapten a distintos entornos de aplicación.

Como un ejemplo de tales operadores particulares podemos considerar el uso de órdenes que indiquen niveles de confiabilidad en la información expresada por los hechos o reglas; un operador con éstas características sería adecuado para su utilización sobre ontologías donde la información codificada en las mismas es obtenida a través de procesos automáticos de diferentes sitios web, en tal entorno podríamos asociar la confianza que se tiene el sitio que es fuente de una información con el dato en cuestión, para que de esta forma el operador de consolidación resuelva dejar aquellas fórmulas a las que mayor confianza se le tiene, mejorando de esta forma la calidad en la información de la ontología consolidada. A su vez, como se ha mostrado en los Ejemplos 5.5 y 6.5, otra opción es utilizar la cantidad de átomos que son perdidos al remover cierta fórmula.

Claramente, distintos entornos de aplicación se beneficiarán de diferentes relaciones (y combinaciones entre las mismas). Es por esto que como continuación del trabajo presentado en esta disertación proponemos analizar diferentes entornos de aplicación de los operadores, de forma que puedan ser definidos los mejores criterios para éstos.

7.3.3. Manejo de incoherencias a través de debilitación de reglas: Defeasible Datalog$^{\pm}$

Por otra parte, una propuesta alternativa para el manejo de incoherencias que pensamos explorar es la basada en debilitamiento de reglas, en lugar de remoción de las mismas. Para lograr esto pensamos explotar unos desarrollos recientes que han surgido dentro del grupo de investigación que anidó el trabajo presentado en esta disertación, pero que se

centran en la modificación de la relación de consecuencia para el manejo de ontologías inconsistentes: Datalog$^{\pm}$ rebatible [MDFS14].

Este formalismo se basa, entre otras cosas, en clasificar las TGDs como rebatibles, lo que hace que las conclusiones de TGDs puedan ser descartadas luego de un proceso dialéctico basado en Argumentación [DW09, Sim89, SL92], el cual sopesa razones a favor o en contra de un literal para decidir si el mismo es respuesta o no del sistema cuando el mismo está en conflicto con otro a través de alguna NC o EGD. Esto nos brinda una posibilidad diferente a la hora de pensar en debilitar reglas; en lugar de utilizar un enfoque de agregar manualmente excepciones como el propuesto por Qi *et al.* en [QLB06] (que, como se explicó anteriormente, involucraría agregar cada individuo como excepción en caso de incoherencia), lo que podemos hacer una vez que hemos identificado un kernel de dependencias es transformar reglas dentro del mismo a rebatible. De esta forma, incluso cuando todavía se podrían derivar átomos que mapeen a la NC o EGD violada en cuestión, los mismos serían átomos rebatibles. Esto haría que, en lugar de producir una violación de la restricción, lo que suceda es que se dispare el proceso dialéctico de forma que podamos definir con cual de los átomos que provocan el conflicto nos quedamos. De esta forma resolveríamos la incoherencia, ya que un conjunto de átomos relevantes no provocaría inevitablemente la violación.

Es más, como podemos ver esto no involucraría agregar como excepción a cada conjunto de átomos relevantes a un kernel de dependencias; en su lugar, simplemente la transformación a versión rebatible de la TGDs alcanzaría para no sólo para resolver las violaciones actuales sino también las futuras (en el enfoque de Qi *et al.* futuras violaciones implican claramente nuevas adiciones de excepciones en futuras iteraciones de los operadores).

7.3.4. Implementación de operadores

Finalmente, en la actualidad estamos trabajando en la implementación de nuestros operadores; planeamos para esto estudiar diferentes técnicas que puedan ser utilizadas para obtener una implementación eficiente de los operadores, posiblemente enfocados en fragmentos específicos de Datalog$^{\pm}$.

En este sentido, el trabajo de Wasserman en [Was00] muestra cómo el uso de funciones de incisión mínimas (tales como las que generan consolidaciones que satisfacen **Mínima**

Pérdida de Información) pueden ser obtenidas a partir de los kernels de una KB mediante el uso de cualquier algoritmo para encontrar minimal hitting sets [Rei87]. Muchos trabajos en el área de reparación de ontologías [Kal06, KPSH05, HK06] han explotado los algoritmos de Reiter para lograr una implementación eficiente de sus frameworks.

Entre otras opciones, planeamos estudiar que tan adecuadas son estas técnicas para nuestros operadores. En este sentido, como dijimos hay una relación prácticamente directa entre funciones de incisión mínimas y los minimal hitting sets de Reiter; sin embargo, la relación entre los últimos y funciones de incisión no minimales ya no es tan directa. Por lo tanto, el enfoque de Reiter puede ser de mucha utilidad a la hora de implementar los operadores globales basados en clusters; para la implementación de operadores locales tanto modificaciones a los algoritmos de búsqueda de minimal hitting sets como otras alternativas serán exploradas.

Capítulo 8

Conclusiones y trabajo relacionado

En la presente disertación hemos presentado operadores para la consolidación de ontologías Datalog$^{\pm}$ que encaran tanto problemas de inconsistencia como incoherencia. Las construcciones mostradas se guían por la noción de mínima pérdida de información, considerando para ello el impacto de la remoción de fórmulas en la ontología. A partir de tal noción hemos identificado dos enfoques distintos, basados en el tratamiento local o global de conflictos mínimos.

En el presente capítulo brindamos las conclusiones más importantes obtenidas durante el desarrollo del presente trabajo, desde un análisis de aquellas características del entorno de aplicación que nos han llevado a identificar y tratar de manera adecuada el fenómeno de incoherencia hasta las razones detrás de la separación en tratamientos locales y globales de conflictos mínimos. Primeramente, de todas formas, presentamos una recapitulación y comparación de trabajos de los campos de Inteligencia Artificial y Bases de Datos que se encuentran relacionados con el presentado aquí, ya sea a nivel objetivos (*v.g.,* el tratamiento de conflictos) o directamente a nivel construcción (por ejemplo, en el uso de algún tipo de medida sobre la base de conocimiento como estrategia de resolución de problemas).

8.1. Trabajos relacionados

En el presente trabajo hemos presentado diferentes operadores de consolidación de ontologías Datalog$^{\pm}$ que apuntan a la resolución de los problemas de incoherencia e inconsistencia en las ontologías. Hasta donde sabemos, no existe en la literatura otro trabajo

que apunte específicamente a la consolidación de ontologías Datalog$^{\pm}$. A continuación presentaremos diversos trabajos que se encargan de resolver de alguna forma incoherencias o inconsistencias, estableciendo comparaciones entre esos enfoques y el presentado en esta disertación. Si bien como dijimos no existen actualmente trabajos enfocados en la consolidación de ontologías Datalog$^{\pm}$, comenzaremos nuestro recorrido por trabajos relacionados estableciendo una comparación con el framework de Lukasiewicz *et al.* [LMS12] para obtención de respuestas desde ontologías Datalog$^{\pm}$ inconsistentes. A continuación, nos enfocaremos en trabajos clásicos del área de Revisión de Creencias que versan sobre bases de conocimiento proposicionales, tanto en revisión de las mismas como integración y actualización. Finalmente, nos centraremos en diversos enfoques que se centran en el manejo de inconsistencias e incoherencias en entornos más cercanos a Datalog$^{\pm}$, a saber Lógicas Descriptivas, Programación Lógica y Bases de Datos Relacionales.

8.1.1. Manejo de inconsistencias en Datalog$^{\pm}$: el framework de Lukasiewicz *et al.*

Quizás el trabajo más íntimamente relacionado con el presentado aquí es el de Lukasiewicz *et al.* [LMS12]. Sin embargo, tal trabajo toma una dirección distinta al nuestro; perteneciendo al grupo de esfuerzos que apuntan a modificar cómo se infiere conocimiento en presencia de inconsistencia en lugar de atacarla directamente como nuestros operadores. En tal trabajo los autores definen un framework general para la resolución de consultas tolerante a inconsistencias en ontologías Datalog$^{\pm}$ que también se basa en la utilización de funciones de incisión. Sin embargo, como anticipamos previamente Lukasiewicz *et al.* se enfocan en forzar consistencia en tiempo de ejecución de la consulta obteniendo respuestas consistentes de una ontología inconsistente en lugar de consolidar la ontología. Claramente, el proceso propuesto por Lukasiewicz *et al.* debe ser llevado a cabo para cada consulta realizada al sistema, mientras que en nuestro enfoque obtenemos una nueva KB de manera *offline*, y el conocimiento puede ser consultado por métodos clásicos sin considerar problemas de consistencia; ambos enfoques pueden ser útiles, dependiendo del entorno de aplicación. Adicionalmente, como sólo una KB es usada la suposición racional de que no hay conflictos en las dependencias y restricciones en Σ es hecha, y por lo tanto no existe la nociones de insatisfacibilidad e incoherencia. Como hemos establecido previamente, para ganar generalidad en la presente tesis hemos decidido desechar tal suposición, y tratamos problemas de incoherencia además de los de inconsistencia.

8.1.2. Manejo de inconsistencias en Lógica Proposicional

Además del trabajo de Lukasiewicz *et al.*, existen muchos otros trabajos que resuelven inconsistencia o incoherencia mediante una adaptación de enfoques basados en técnicas de Revisión de Creencias en otros formalismos de representación de conocimiento. Principalmente, existen numerosos trabajos enfocados en la revisión e integración de bases de conocimiento proposicionales (por ejemplo [KP02, KM92, LM99, LS98, EKM08, KP11, DDL06, BMVW10, Del11, DJ12, FKIRS12]), los cuales han provisto las bases para otros trabajos en (fragmentos de) lógicas de primer orden. Como es de esperarse, esos trabajos tienen conexiones profundas con nuestro enfoque, pero a su vez remarcables diferencias, como veremos a continuación.

Como hemos mencionado previamente en la presente tesis, el trabajo de Sven Ove Hansson en [Han94] proveyó la inspiración y las bases para nuestro trabajo: seguimos un enfoque semejante a Kernel Contraction y varias intuiciones del mismo, adaptadas a un lenguaje ontológico como Datalog$^{\pm}$. En consecuencia, además del tratamiento de incoherencia proveemos un proceso de resolución de inconsistencias completo, el cual esta basado en el tratamiento de conjuntos conflictivos mínimos, atacados por funciones de incisión similares en espíritu a las propuestas por Hansson. Sin embargo, en este trabajo en lugar de utilizar funciones de incisión generales como las definidas por Hansson (las cuales no establecen como las mismas eligen que fórmula remover) se utiliza una relación de preferencia entre las fórmulas para establecer que debemos remover. En base a tal relación presentamos dos enfoques, que se diferencian principalmente en que mientras uno trata cada conflicto de manera local, el otro se enfoca en la interacción entre los distintos kernels (cuando la misma existe) a través del uso de una nueva estructura denominada clusters. Hemos mostrado en la Sección 6.3, principalmente en los ejemplos 6.3 y 6.4, que el enfoque global de Cluster Contraction no se corresponde con el enfoque local de tratamiento de conflictos, teniendo los mismos características y comportamientos diferentes al momento de consolidar una ontología Datalog$^{\pm}$. Sin embargo, cabe aclarar que ambos enfoques son en realidad particularizaciones del enfoque propuesto por Hansson; esto es, tanto el enfoque local como el enfoque global pueden ser obtenidos a través de las funciones de incisión generales presentadas en [Han94], adaptando las mismas a ontologías Datalog$^{\pm}$.

Konieczny y Pino-Pérez en [KP02] realizaron una de las más importantes contribuciones en la integración de información conflictiva. En nuestro trabajo seguimos algunas intuiciones propuestas por estos autores. Sin embargo, la principal diferencia entre su en-

foque y el nuestro (dejando de lado la diferencia obvia entre los objetivos de los trabajos, consolidación contra integración) es que ellos determinan que la ontología obtenida de la integración será consistente sólo en caso de que que el conjunto de restricciones de integridad usado para guiar la integración sea consistente (o, en nuestra terminología, coherente), y no analizan el caso alternativo.

Respecto de [LM99], además de disentir en el foco del trabajo (nuevamente, integración contra consolidación), la principal diferencia en la estrategia de manejo de inconsistencia con nuestro trabajo es que su estrategia de resolución de conflictos se basa en los "votos" de la mayoría para establecer qué fórmula es retenida en la integración. En lugar de eso, nuestra estrategia se basa en la pérdida de información inducida para la remoción de fórmulas. Ninguno de los dos criterios puede ser considerado mejor que el otro en nuestro entorno de ontologías Datalog$^{\pm}$. Como suele ocurrir en muchos enfoques de resolución de conflictos, la elección de una estrategia específica depende fuertemente del entorno de aplicación y los usuarios.

Katsuno y Mendelzon [KM92] atacan el problema de la revisión de bases de conocimiento proposicionales. Al igual que en nuestro trabajo, el mismo lenguaje es utilizado para expresar tanto los hechos conocidos del mundo como las restricciones impuestas a tal conocimiento. Sin embargo, nuevamente la diferencia entre la actualización (en este caso) de una *KB* y su consolidación a través de los operadores presentados en el presente trabajo surge del tratamiento que le dan Katsuno y Mendelzon a las restricciones de integridad: en su trabajo los autores consideran a las restricciones de integridad como invariantes y las actualizaciones para restaurar consistencia son restringidas a los hechos. Como hemos explicado anteriormente, en este trabajo hemos elegido considerar que las restricciones pueden cambiar con el tiempo, lo que termina significando que incoherencias pueden aparecer en las mismas. Por lo tanto, además de modificar los datos, nuestros operadores modifican también el conjunto de restricciones de la base de conocimiento.

En [Del11, DJ12] los autores presentan un enfoque para la revisión de una base de conocimiento proposicional donde la entrada epistémica es un conjunto de sentencias, en lugar de una sola sentencia como es el caso en la gran mayoría de los trabajos clásicos de revisión. Este conjunto de sentencias es tal que cada sentencia en el puede ser aceptada de manera independiente al considerarla con la *KB* original, pero puede haber inconsistencias cuando se consideran todas las sentencias de la entrada epistémica en conjunto. La idea principal del enfoque de los autores sigue las intuiciones de la teoría AGM, pero

difiere en que es necesario alterar el postulado de Éxito de AGM para que se acomode a la intuición de que no toda fórmula en el conjunto debe encontrarse presente en la base de conocimiento revisada final (ya que como se dijo el conjunto de sentencias que es entrada epistémica puede ser inconsistente). Guiados por el principio de mínima pérdida, los autores caracterizan a la revisión como los mundos mas plausibles entre los varios subconjuntos maximalmente consistentes del conjunto de sentencias original. Estableciendo un paralelo con nuestro entorno de aplicación de ontologías Datalog$^{\pm}$, este enfoque de revisar el conjunto de sentencias podría emparentarse con la revisión del componente D de la ontología para resolver inconsistencias; siendo el conjunto de sentencias inconsistente, la unión del mismo con la KB original será también inconsistente. No obstante, hay una diferencia importante entre tales trabajos y el nuestro. En esos trabajos los autores primero resuelven las inconsistencias en el conjunto de sentencias, de forma tal que puedan decidir cual de los subconjuntos maximalmente consistentes del mismo caracterizará la revisión. Nuestro enfoque es distinto, ya que nosotros directamente consideramos una KB (potencialmente) inconsistente. De esta forma, para resolver el mismo problema en nuestro entorno de aplicación deberíamos considerar directamente la unión de la KB con la totalidad del conjunto de sentencias que es entrada epistémica, y recién en ese momento aplicar el operador de consolidación para resolver problemas de consistencia si los hubiera.

8.1.3. Manejo de inconsistencias e incoherencias en entornos ontológicos: Description Logics, Logic Programming y Bases de Datos Relacionales

Ahora nos enfocamos en otros formalismos de representación de conocimiento más cercanos a Datalog$^{\pm}$, principalmente en la familia de Lógicas Descriptivas [BCM^{+}03] y Programación Lógica [Llo87, NM95, Gel08]. Compararemos con nuestra propuesta diferentes técnicas de manejo de inconsistencias para ontologías DL y programas lógicos, así como también otras técnicas que han tenido sus raíces en el área de bases de datos. A continuación recapitularemos brevemente los trabajos, a la vez que procedemos a establecer similitudes o diferencias con nuestra propuesta.

Como hemos dicho anteriormente, uno de los trabajos más destacables donde se utilizan técnicas de Revisión de Creencias para la resolución de conflictos en ontologías DL es el de Qi *et al.* [QLB06], el cual está basado en la teoría AGM [AGM85, Gär88]. Lo que

destaca a este trabajo es que en él los autores introducen generalizaciones de los postulados AGM para el caso de DL al tiempo que introducen dos operadores sobre bases de conocimiento, basados en debilitamiento de fórmulas (*esto es,* la adición de excepciones a reglas). Esta técnica de debilitamiento se basa en añadir como excepciones a una regla de conocimiento aquellos individuos que generan conflictos de consistencia. Sin embargo, es importante destacar que los operadores en [QLB06] no proveen tratamiento de incoherencias. Como hemos visto en el capítulo 3, hay una estrecha relación entre los conceptos de incoherencia e inconsistencia. Tal como muestra la Proposición 3.1, un conjunto de átomos que sea relevante a un conjunto insatisfacible de TGDs provoca inevitablemente una inconsistencia. Trasladando este resultado a ontologías DL esto significa que cada individuo que pertenece al concepto insatisfacible en cuestión genera un conflicto de consistencia en la ontología. Por lo tanto, en la práctica esto puede provocar que en presencia de incoherencia, si no manejamos la misma explícitamente entonces los operadores propuestos por Qi *et al.* realicen debilitamientos extremos donde cada individuo sea anotado como una excepción. Como hemos visto en los capítulos 5 y 6, los operadores de consolidación propuestos en la presente tesis proveen no sólo tratamiento de inconsistencias sino también de incoherencias, evitando este tipo de situaciones. De todas formas, esto no significa que los enfoques sean opuestos. Es sencillo ver que los mismos podrían complementarse fácilmente; por ejemplo, si modificamos los operadores definidos en las definiciones 5.6 y 6.6 para que solamente ejecuten la primer parte de su accionar (esto es, la resolución de incoherencias) entonces podríamos tomar una KB inconsistente e incoherente, y aplicar sucesivamente primero nuestros operadores modificados y luego sobre la KB coherente resultante los operadores de Qi *et al.* para realizar los debilitamientos pertinentes que resuelvan las inconsistencias.

Otro trabajo importante que hemos presentado previamente es el de Meyer *et al.* [MLB05], el cual adapta dos métodos clásicos de revisión en Lógica Proposicional a DL, utilizando para ellos bases de conocimiento disyuntivas (*esto es,* el resultado es una disyunción de fórmulas), que como hemos explicado previamente tiene sus desventajas a la hora de ser explotadas por procesos de razonamiento. En los capítulos 5 y 6 se introdujeron operadores de consolidación que difieren del propuesto por Meyer *et al.* en dos aspectos cruciales además del formalismo de representación de conocimiento abordado. Por un lado, en la consolidación de ontologiás Datalog$^{\pm}$ obtenemos una ontología Datalog$^{\pm}$ tradicional que puede ser explotado por cualquier método de razonamiento sobre Datalog$^{\pm}$ favoreciendo la compatibilidad, algo que no siempre es posible con DKBs.

Por el otro, si bien Meyer *et al.* reconocen a la incoherencia como un problema importante, en sus operadores ellos deciden no atacar problemas de incoherencia, enfocándose exclusivamente en inconsistencias; aclarando que dejan los problemas de consistencia para investigaciones futuras.

Un enfoque diferente (incluido dentro del grupo de esfuerzos que modifican la relación de consecuencia en lugar de modificar las bases de conocimiento conflictivas) es el propuesto en [BHP09], el cual es capaz de usar información proveniente de varias ontologías DL a la hora de resolver una consulta, considerando en el proceso conflictos tanto de consistencia como de coherencia. El enfoque propuesto por Black *et al.* está basado en el uso de agentes con capacidades argumentativas, cada uno con una base de conocimiento personal en forma de una ontología DL. Estos agentes utilizan juegos de diálogo para intercambiar argumentos (*esto es,* razones a favor o en contra de una conclusión, que es una posible respuesta a una consulta) hasta que los mismos llegan a un acuerdo acerca de la respuesta a una consulta. Por lo tanto, los agentes pueden usar la (posiblemente incoherente/inconsistente) unión de las ontologías sin la necesidad de integrar las mismas, obteniendo de todas formas una respuesta influenciada por el conocimiento de todas las ontologías en juego (*esto es,* que considera el conocimiento de cada agente). Es más, este enfoque tiene la ventaja de que, como ninguna formula es removida de las ontologías, no se produce pérdida de información. Como resultado de esto, las conclusiones que se pueden obtener de mediante este enfoque son un superconjunto de las que pueden obtenerse de la ontología resultante de la consolidación de la unión de las ontologías DL de los agentes. Sin embargo, incluso cuando los autores claman que una ventaja de su enfoque es que ellos no necesitan gastar tiempo y esfuerzo en realizar la consolidación de la KB, la realidad es que una gran desventaja de cualquier método que involucra el uso de razonamiento argumentativo es la complejidad computacional asociada al mismo [PWA03, DW09, CFS06]. Esto se vuelve más importante al considerar que el diálogo argumentativo debe llevarse a cabo para cada una de las consultas realizadas al sistema, de forma concurrente o *online*. Incluso cuando un proceso de consolidación puede también ser computacionalmente costoso, sólo es necesario realizarlo una vez, y ésto puede realizarse *offline* antes de que el sistema de resolución de consultas sea puesto a disponibilidad para su uso. Es decir, ambos enfoques tienen sus méritos, que los hacen atractivos para diferentes tareas. La elección de un enfoque sobre el otro dependerá fuertemente en el entorno de aplicación en el que será utilizado, *esto es,* en el tamaño de las ontologías que serán utilizadas, qué tan frecuentes serán actualizadas las KBs o que tan crítico es el consumo de tiempo para el

sistema, entre otras consideraciones; por supuesto, el conjunto de inferencias que pueden ser obtenidos por cada enfoque difiere y esto también debería ser considerado. Un enfoque basado en consolidación podría ser más adecuado para sistemas dependientes del tiempo como ser sistemas de tiempo real o sistemas de consultas intensivas; en tales entornos la tratabilidad en datos asociada con (la consolidación de) una ontología Datalog$^{\pm}$ probablemente demuestren ser muy útiles.

Otro área que ha tenido una gran expansión en los últimos tiempos que la ha llevado indefectiblemente a enfocarse en problemas de inconsistencia es el de acceso a datos basado en ontologías (ontology-based data access - OBDA). El trabajo en [LLR$^+$10] estudia la adaptación para ontologías DL-Lite de un enfoque proveniente del área de base de datos conocido como Resolución Consistente de Consultas (Consistent Query Answering - CQA), el cual se basa en la utilización de *repairs*, que son conjuntos de átomos que satisfacen un conjunto de restricciones al tiempo que están "tan cerca como es posible" a la base de datos original. La adaptación propuesta, denominada semántica AR (ABox repair semantics), se basa en aquellos átomos que pueden ser inferidos de los repairs de la ontología original. A su vez, un refinamiento es propuesto conocido como semántica de intersección (IAR semantics) el cual es una aproximación sensata de respuestas consistentes a AR; esta semántica consiste en computar la intersección de todos los repairs posibles y obtener las respuestas de tal intersección. Aunque (posiblemente muchas) respuestas de AR pueden ser perdidas bajo la semántica IAR, éstas últimas son más sencillas de obtener desde un punto de vista computacional para la familia DL-Lite, ya que no es necesario computar todo el conjunto de repairs para obtener su intersección. Las complejidades tanto en datos como combinada de estas y otras semánticas fueron estudiadas en [Ros11] para un gran espectro de Lógicas Descriptivas. Además, en [Ros11] resultados de intratabilidad para la resolución de consultas fueron encontrados para $\mathcal{EL}_\bot$ bajo la semántica de intersección, y se pudo comprobar que un segmento no recursivo del lenguaje es computable en tiempo polinomial. Más cerca en el tiempo, en [BR13], otra familia de aproximaciones a CQA fueron propuestas, también con la familia de DL-Lite como foco de los esfuerzos. La semántica de k-soporte (k-support semantics) permite aproximar (de manera sensata) el conjunto de consultas inferido bajo la semantica CQA, basándose en los k subconjuntos de la base de datos que infieren q (la consulta) de manera consistente; por el otro lado, la semántica de k-derrotadores (k-defeater) es una aproximación completa de CQA que busca conjuntos que contradicen a los soportes de q. De manera similar a [BHP09], el tratamiento de inconsistencias propuesto por todas estas semánticas está relacionado con

consultas particulares en lugar de tratar la inconsistencia de la ontología en conjunto. Por lo tanto, ellos no intentan obtener una base de conocimiento final consistente que pueda ser consultada sin restricciones, *esto es,* se encuadran dentro del grupo de esfuerzos que modifican como las respuestas son obtenidas. Es más, al contrario de nuestra propuesta ellos no encaran el problema de incoherencia y el de inconsistencia al mismo tiempo pero por separado (*esto es,* primero incoherencia y luego inconsistencia). En lugar de eso, los enfoques o bien asumen que las restricciones de integridad definen de forma correcta la semántica de la instancia de base de datos, por lo que no hay lugar para incoherencia; o bien tratan inconsistencias e incoherencias de la misma forma al momento de remover o ignorar información, lo que puede llevar a los problemas que hemos mostrado en el Ejemplo 5.4. Si bien estas técnicas pueden ser adecuadas para el caso de bases de conocimiento coherentes, en presencia de incoherencia en el conjunto de restricciones de integridad estos enfoques pueden llegar a producir conjuntos vacíos de respuestas, ya que ningún subconjunto de la base de datos puede satisfacer las restricciones — como también puede suceder en el caso del enfoque propuesto en [LMS12].

Centrándonos en Programación Lógica, hay muchos trabajos que encaran el problema de integrar bases de conocimiento expresadas como programas lógicos, resolviendo en el proceso problemas de inconsistencia. Como explicamos en la Sección 4.6, uno de estos trabajos es el de [HPW09], donde los autores realizan procesos de integración basándose en la semántica de modelo estable. La estrategia de integración propuesta por Hué *et al.* utiliza pre-órdenes entre los candidatos a remover para de esta forma determinar cuales de los candidatos son efectivamente removidos. Como hemos mostrado en el Capítulo 5, en la presente tesis utilizamos un enfoque similar basado en órdenes generales entre las fórmulas en las ontologías Datalog$^{\pm}$, de forma tal que estos ordenes puedan particularizarse para adaptarse a las necesidades del entorno de aplicación particular en el cual los operadores serán utilizados, por ejemplo, en la consideración de la pérdida global de información ocurrida cuando las fórmulas son eliminadas. Compartiendo este razonamiento, Hué *et al.* no proveen una forma de obtener los preórdenes a utilizar: en su lugar, los autores asumen que para cualquier estrategia P hay un pre-orden dado que la define. Sin embargo, también hay diferencias notorias entre los trabajos, siendo la más importante de ellas el foco de los mismos; mientras que Hué *et al.* encaran el problema de la integración de programas lógicos (que, por citar una diferencia, responden a una suposición de mundo cerrado) aquí nos encargamos de la consolidación de conocimiento expresado a través de ontologías Datalog$^{\pm}$ (suposición de mundo abierto, incoherencias además de inconsistencias, entre

otras características diferentes).

Como explicamos anteriormente en la Sección 4.6, Delgrande *et al.* presentan en [DSTW09] otro notorio trabajo de integración en Programación Lógica que involucra como es de esperarse manejo de inconsistencias. En ese trabajo se presentan dos enfoques distintos: arbitración por un lado (*esto es,* intuitivamente, dando la misma importancia a cada programa), y el uso de un programa especial como modelo y aproximación al mismo por el otro. El primer enfoque está muy relacionado con el que proponemos en esta tesis ya que la medida de distancia usada en él es definida en una manera similar a la propuesta aquí (utilizando la diferencia simétrica entre interpretaciones), y además porque el caso de programas insatisfacibles es estudiado, de forma similar a cómo en esta tesis se estudia el problema de incoherencias en Datalog$^{\pm}$, reconociendo de esta manera la importancia de tales conflictos. Sin embargo, ellas consideran la satisfacibilidad o no del programa en su totalidad, y no de un subconjunto de reglas o un concepto particular. Además, como se explico anteriormente, debido a la diferencia en los objetivos de ese trabajo con respecto a la presente tesis la estrategia utilizada para el manejo de incoherencias es totalmente diferente: en lugar de modificar el conocimiento para recuperar coherencia, Delgrande *et al.* ante la presencia de un programa incoherente directamente eligen dejarlo fuera de consideración en la integración.

En el área de base de datos, uno de los trabajos más influyentes es el de Arenas *et al.* [ABC99] en resolución consistente de consultas (Consistent Query Answering - CQA), en el cual los autores proponen una definición de respuestas consientes a nivel de modelo teórico para una consulta a una base de datos relacional que puede estar en estado de inconsistencia con respecto a restricciones de integridad. Intuitivamente, las respuestas consistentes a una consulta es el conjunto de átomos que son respuestas (clásicas) a consultas en *todos* los repairs de la base de datos inconsistente. Como hemos explicado previamente, un repair es un conjunto maximal (tan cerca como sea posible de la base de datos original) de átomos que satisface las restricciones impuestas a los mismos. Diferentes nociones de repair han sido estudiadas en la literatura, así como diferentes nociones de qué significa que un conjunto de átomos esté tan cerca como es posible de la base de datos original. La mayoría de las propuestas están basadas en reparar mediante inserción y/o remoción de tuplas a/de la base de datos (donde las acciones posibles a realizar dependen de la forma de las restricciones de integridad y su expresividad) y la noción de cercanía es definida mediante inclusión conjuntista o cardinalidad. El trabajo en [ADB07],

sin embargo, propone un framework uniforme para representar e implementar distintos enfoques para la reparación de bases de datos basándose en minimizar distancias que son dependientes del dominio de aplicación. La idea principal en ese trabajo es mostrar cómo pensar en términos de diferentes distancias para expresar preferencias entre los repairs lleva a diferentes estrategias que pueden ser aplicadas en distintos escenarios. Los autores muestran que el conjunto de repairs ontenido usando las funciones de distancia propuestas difieren de aquellos que se obtienen al considerar inclusión conjuntista. Es más, además de la inserción y remoción de tuplas completas, hay varios otros enfoques dependientes del dominio que se basan en cardinalidad y otras funciones objetivo más complejos. En el caso del enfoque propuesto por Wijsen [Wij05] las actualizaciones son considerados como una primitiva en el framework teórico desarrollado; Bohannon *et al.* en [BFFR05] presentan un framework basado en costo que permite hallar "buenos" repairs para bases de datos que exhiben inconsistencias en forma de violaciones a dependencias funcionales o de inclusión, permitiendo actualizaciones a valores de atributos. En ese trabajo, dos heurísticas son definidas para construir repairs basándose en clases de equivalencias de valores de atributos (donde los algoritmos presentados se basan en una selección de la reparación menos costosa). A su vez, los autores exploran diversas optimizaciones de performance. Una semántica totalmente diferente para la reparación de bases de datos es la propuesta en [CGZ09, CT11] a través del uso de restricciones de integridad activas (Active Integrity Constraints - AICs); una AIC es una regla de producción donde el cuerpo es una conjunción de literales, que debe ser evaluada como falso para que la base de datos sea consistente, mientras que la cabeza es una disyunción de átomos de actualizaciones que deben ser ejecutadas si el cuerpo es evaluado como verdadero (es decir, si la restricción es violada y por lo tanto la base de datos es inconsistente). Entonces, los repairs son definidos como un conunto mínimal de acciones de actualización (inserción /remoción de tuplas) y AICs especifican las acciones usadas para recuperar consistencia. Por lo tanto, de todos los posibles repairs, sólo el subconjunto de repairs consistente de actualizaciones soportadas por las AICs son consideradas. Finalmente, otros trabajos en el área de base de datos proponen diferentes semánticas mediante la consideración explícita o implícita de una relación de preferencia entre el conjunto de (*v.g.,* [AFM06, SCM12, GM12]). Como ocurre a menudo, la gran diferencia entre estos trabajos y el propuesto en esta tesis es que los mismos no consideran problemas potenciales a partir de conjuntos de restricciones incoherentes, por lo que no se proveen en ellos tratamiento de inconsistencias alguno, que cómo hemos visto puede acarrear graves problemas a la hora de encontrar los repairs, al

punto límite de no poder brindar ninguna respuesta útil. Nuevamente, esto no significa que los enfoques sean contrarios, sino que brinda una gran oportunidad de interacción donde los operadores pueden combinarse de manera apropiada para que unos se encarguen del manejo de incoherencias y de esta forma dejar cumplidas las pre-condiciones esperadas para la aplicación de los otros.

También relacionado con el área de base de datos es el enfoque propuesto en [LM98], aunque el mismo es totalmente diferente a aquellos que hemos tratado hasta el momento. En ese trabajo, una base de datos es modelada como una teoría de primer orden sin reglas, y las restricciones de integridad son usadas para asegurar la consistencia en el resultado final, en un paralelo a lo realizado por Konieczny y Pino-Pérez en [KP02]. En el trabajo se presentan formas de resolver problemas conocidos de la integración de bases de datos como los sinónimos (campos en distintas bases de datos representan el mismo objeto real pero tienen nombres diferentes) y los homónimos (diferentes implementaciones de la misma relación). Sin embargo, como en el caso del trabajo de Konieczny y Pino-Pérez, los autores no consideran problemas relacionados con el conjunto de restricciones de integridad. En lugar de eso, el conjunto de restricciones usados para hacer la integración es único, y la elección de tal conjunto se espera que sea realizada por un diseñador de integración. En oposición a las decisiones de diseño en [LM98], nosotros no establecemos restricciones en la forma de las restricciones para nuestro entorno de consolidación de ontologías Datalog$^{\pm}$.

En [Cho98], Cholvy introduce otro enfoque para razonar con información contradictoria. El framework es representado a través de un conjunto de axiomas y reglas de inferencia. En adición, en el trabajo varias aplicaciones para el framework son introducidas, *v.g.,* la resolución de conflictos entre creencias representadas por bases de datos de primer orden, donde los hechos son literales fijos y hay reglas que pueden ser restricciones de integridad o reglas de deducción. En ese escenario, la contradicción es detectada mediante la aplicación de las restricciones cuando se consideran varias bases de datos en conjunto. Esto establece un cierto paralelo con el caso de inconsistencia en una ontología Datalog$^{\pm}$. Sin embargo, la diferencia principal con nuestro trabajo radica en cómo la estrategia para el manejo de inconsistencia es definida. En ese trabajo, se asume la existencia de un orden de preferencia entre las bases de datos en juego, el cual es utilizado como base de la definición de la estrategia. En lugar de esto, en esta disertación definimos un orden entre las fórmulas basado en la información contenida en la ontología original. Sin embargo, es

importante aclarar que claramente en nuestro framework el criterio para definir el orden es modular, y por lo tanto puede ser cambiado acorde, por ejemplo estableciando un orden arbitrario, obteniendo de esta forma una estrategia de manejo de inconsistencia similar a la introducida por Cholvy.

Finalmente, varios enfoques de manejo de inconsistencia son introducidos en [HvHH+05]. Haase *et al.* se enfocan en las bases formales comunes a todos los enfoques, para de esta forma poder compararlos y caracterizar escenarios donde un enfoque es más útil que el resto; dentro de los enfoques estudiados el más relacionado a nuestro trabajo es el de reparación de inconsistencias. Para lograr esto, este enfoque primeramente localiza las inconsistencias mediante la obtención de subconjuntos inconsistentes de la ontología y luego aquellos mínimos son computados, *esto es,* claramente esto es lo que realizamos en nuestro framework al obtener los kernels de datos; luego, los autores pasan a la resolución de la inconsistencia. La estrategia básica propuesta es similar a la que proponemos en esta disertación: remover axiomas para hacer el conjunto resultante consistente. Sin embargo, es importante aclarar que los autores no introducen ninguna forma particular de seleccionar qué axiomas remover; es más, ellos claman que la elección es usualmente dejada al usuario final. Por el contrario, en la presente disertación presentamos una construcción completa de operadores de consolidación donde la elección de las fórmulas a remover es realizada de forma automática, sin tener dependencia del usuario.

8.2. Conclusiones

El trabajo colaborativo y el intercambio de información se han transformado en los últimos tiempos en aspectos cruciales de prácticamente todo sistema; por lo tanto, es de vital importancia disponer de métodos automáticos y adecuados para el manejo de conflictos: en entornos colaborativos, a medida que el conocimiento evoluciona inconsistencias e incoherencias suelen aparecer de manera natural. Es más, en entornos colaborativos tal conocimiento es frecuentemente representado a través de ontologías que suelen ser mantenidas en forma conjunta por muchas entidades, *esto es,* son compartidas por entidades que no sólo las utilizan sino que también las modifican.

Una forma de lidiar con los conflictos que pueden aparecer en tales entornos de aplicación es tratar de modificar la información contenida en la ontología para de esta forma recuperar la coherencia y la consistencia del conocimiento expresado por las mismas.

En este trabajo hemos presentado una forma de conseguir la consolidación de ontologías Datalog$^{\pm}$, *esto es,* la resolución de todos los conflictos presentes en las mismas, tanto de inconsistencia como de incoherencia.

En particular, el fenómeno de incoherencia es uno que suele ser dejado de lado, a pesar de tener gran influencia en la definición de procesos de manejo de inconsistencia. Como fue explicado en el Capítulo 3, en lugar de atacar los problemas de coherencia en el conocimiento muchos enfoques clásicos de manejo de inconsistencia optan por asumir que las restricciones que utilizan para guiar el proceso de manejo de inconsistencia no tienen problemas y por lo tanto son satisfacibles. Un ejemplo de este enfoque es el trabajo seminal de Konieczny y Pino-Pérez en integración de creencias [KP02] (donde se toman varias bases de conocimiento y se intenta integrarlas de manera consistente). Otros ejemplos son trabajos como [ABC99, LLR^{+}10, LMS12], donde se asume que el conjunto Σ expresa la semántica de los datos en la base de datos D. De la mano de tal suposición los autores asumen que no hay ningún conflicto interno en las restricciones; es decir, se asume que las restricciones son satisfacibles en el sentido de que su aplicación no provoca *inevitablemente* un problema de consistencia. Sin embargo, tales suposiciones no siempre son aplicables. En la presente tesis elegimos considerar que tanto los datos como las restricciones pueden cambiar a medida que una ontología Datalog$^{\pm}$ crece, y por lo tanto no sólo consideramos problemas relacionados a los datos sino también a las restricciones impuestas sobre los mismos. Para tal fin hemos presentado una caracterización del fenómeno de incoherencia en Datalog$^{\pm}$, tomando como punto de partida esfuerzos similares en DLs, donde se relaciona a tal fenómeno con la aparición de conceptos insatisfacibles [FHP^{+}06, Bor95, BB97, KPSH05, SHCvH07, QH07]. De forma análoga, en el Capitulo 3 del presente trabajo relacionamos la incoherencia de una ontolgía Datalog$^{\pm}$ en términos de la satisfacibilidad de conjuntos de TGDs con respecto al conjunto de NCs y EGDs en la ontología. Intuitivamente, un conjunto de TGDs es satisfacible si podemos activar todas las reglas en él sin violar ninguna NC ni EGD; y de la mano de esto una ontología es coherente cuando no tiene en su conjunto de TGDs un conjunto insatisfacible.

Además de introducir los conceptos de satisfacibilidad y coherencia para Datalog$^{\pm}$, en el presente trabajo hemos ahondado en la relación entra inconsistencia e incoherencia. En particular, la Proposición 3.1 muestra que tal relación toma la forma de causa-efecto: cuando tenemos una ontología incoherente la misma es inevitablemente inconsistente cuando el componente D hace que un conjunto insatisfacible sea activado. Esta relación es de

vital importancia para la definición de procesos de manejo de inconsistencia: es fácil demostrar que en presencia de una ontología incoherente procesos clásicos de manejo de inconsistencia (*v.g.,* [ABC99, LLR$^+$10, LMS12]) fallan terriblemente, al punto límite de posiblemente no entregar ninguna respuesta. Como podemos ver en el Ejemplo 5.4, en tales casos puede ser una mejor solución aplicar un enfoque de eliminación de incoherencia como paso previo al manejo de inconsistencia, lo que permitirá aumentar la cantidad de respuestas posibles para la ontología previamente incoherente. En particular, esto se hace más evidente considerando aquellos átomos que activan incoherencias por si mismos: de la Proposición 3.1 podemos inferir que tales átomos nunca podrán ser parte de una ontología obtenida por un proceso que sólo involucre el manejo de inconsistencia, ya que los mismos cumplen con los requisitos de la proposición y por lo tanto vuelven a la ontología inconsistente.

De la mano de diferenciar y atacar de forma correspondiente cada tipo de conflicto es que en la presente tesis hemos propuesto la utilización de operadores de consolidación de ontologías Datalog$^\pm$ que tengan en cuenta ambas clases de conflictos, *esto es,* que ataquen tanto incoherencia como inconsistencia. Los operadores propuestos se basan en remover TGDs (para resolver incoherencias) y átomos (en resolución de inconsistencias). Tales remociones se hacen de conjuntos conflictivos mínimos, conocidos como kernels, de forma tal que una vez realizada la remoción el conflicto queda resuelto, mediante la utilización de funciones de incisión, inspiradas en el desarrollo propuesto por Hansson [Han94, Han97b, Han01]. En lugar de la utilización de una función de incisión general que remueva tanto TGDs como átomos, en el presente trabajo hemos elegido separar ambos aspectos y definir paralelamente *funciones de incisión en restricciones* y *funciones de incisión en datos* para atacar adecuadamente cada tipo de conflicto. Durante la presente disertación hemos destacado la importancia de diferenciar, y manejar de manera apropiada, incoherencia de inconsistencia, ya que la calidad de la ontología consolidada dependerá en gran medida de ello, cuando la mínima pérdida de información es aquello que utilizamos como guía en el proceso de consolidación. Esto es particularmente importante a la hora de definir que constituye los kernels y los ordenes entre átomos o restricciones. Como hemos mostrado en la presente tesis (Sección 5.2.3 en general, y Ejemplo 5.4 en particular), al encontrarnos con ontologías incoherentes podemos obtener un mejor comportamiento si computamos los kernels de datos utilizando solamente la parte satisfacible que es retenida del componente Σ, y además especificar el orden entre conjunto de átomos también basándonos en esto.

Esto último explicado es central dentro de la construcción de operadores de consolidación propuesta en la presente tesis, ya que la misma esta basada en la separación en dos tipos de funciones de incisión de modo que se pueda atacar primeramente los problemas de coherencia utilizando una función de incisión en restricciones ρ, y luego resolver los problemas de consistencia basándonos en el resultado de remover de la ontología original aquellas TGDs seleccionadas por ρ. Esto es; el resultado de una operación de consolidación $\Upsilon_{\rho,\varrho}(KB)$ es la ontología obtenida removiendo TGDs (seleccionadas por una función de incisión en restricciones ρ del conjunto de kernels de dependencias $\perp\!\!\!\perp_{(\Sigma,KB)}$) y átomos (seleccionados por una función de incisión en datos ϱ de $\perp\!\!\!\perp_{(D,KB^\star)}$, que es el conjunto de kernels de datos para la ontología resultante de resolver las incoherencias como primer paso) de la ontología original KB. Es importante remarcar que, por un lado, sólo TGDs son removidas de Σ, ya que los kernels de dependencias no contienen EGDs ni NCs. Por el otro lado, como la función de incisión en datos usa $KB^\star$ en lugar de KB entonces sólo átomos de D que estén en conflicto con $\Sigma \setminus \rho(KB)$ son removidos, ya que los kernels de datos son calculados en base a las restricciones obtenidas luego de aplicar sobre Σ la función de incisión en restricciones.

La construcción que ataca directamente kernels de manera local (esto es, la referenciada en el párrafo anterior) tiene un problema al examinarla a la luz de la mínima pérdida de información; en ocasiones, cuando dos o más kernels están relacionados su tratamiento local no es óptimo, como fue mostrado en la Sección 6.1.1. Para paliar tal problema es que hemos definido una estructura adicional denominada cluster, que tiene la función de agrupar kernels relacionados de modo que podamos resolver tales conflictos utilizando un enfoque global del problema. Sin embargo, la aplicación de funciones de incisión definidas para trabajar en kernels directamente a clusters no fue posible, debido a la posibilidad de situaciones anómalas indeseadas (Ejemplo 6.2). A partir de esto se presentan nuevas funciones de incisión adaptadas a su uso en clusters, y se define una nueva clase de operadores de consolidación que aseguran que siempre la ontología resultante de la consolidación es la más cercana a la ontología original (en términos de pérdida de información), aunque como contrapartida el esfuerzo computacional puede llegar a ser mayor.

Paralelamente a la presentación de las construcciones antes mencionadas, en la presente tesis hemos presentado una serie de propiedades que operadores de consolidación de ontologías Datalog$^\pm$ deberían respetar para exhibir un comportamiento que preserve los requisitos de resolver los conflictos que aparecen en las ontologías, tratando a su vez de

minimizar las pérdida de información. Como es característico en Revisión de Creencias, estas propiedades son presentadas como postulados, donde los mismos son independientes de la construcción propuesta; esto es, los postulados pueden ser utilizados para medir el desempeño de cualquier operador de consolidación de ontologías Datalog$^{\pm}$, y no solamente aquellos propuestos en este trabajo. Los postulados propuestos tratan de capturar la noción de que los cambios hechos en la ontología resultante respecto de la original son necesarios, y que la ontología resultante es, como se espera, tanto coherente como consistente. Esto es, dada la ontología original, el proceso de consolidación sólo eliminará restricciones (TGDs) y átomos si los mismos están de alguna forma involucrados en hacer a la ontología original incoherente/inconsistente.

Los postulados presentados versan en distintas direcciones, todas ellas atacando o bien un aspecto relacionado a incoherencia o uno a inconsistencia, o a ambos. Debido a esto es que la gran mayoría de las propiedades actúan indirectamente en una adecuada resolución de conflictos: por ejemplo, requiriendo que las fórmulas que no están involucradas en conflictos queden intactas luego de la consolidación (postulados de **Vacuidad**) o que toda fórmula perteneciente a un conflicto que no sea retenida en el proceso de consolidación sea menos preferida (bajo una relación de preferencia que querramos utilizar) a aquellas del conflicto que fueron retenidas (postulados de **Optimalidad**).

En el presente trabajo hemos también diferenciado aquellas propiedades básicas de las que se obtienen por "herencia" al cumplir con las primeras. Al respecto es interesante remarcar que, aunque quizás pueda parecer extraño a primera vista, las propiedades de coherencia y consistencia se encuentran dentro del grupo de postulados que se infieren de los básicos. Sin embargo, al analizar la situación es natural que esto ocurra: las propiedades de coherencia y consistencia son un fin de los operadores de consolidación, que pueden ser obtenidos a través de diferentes medios. En el caso del conjunto de propiedades propuesto en este trabajo, tales medios son asegurar que sólo se incluye en la consolidación información que previamente se encontraba en la ontología original (**Inclusión**) y que se removió cada conflicto en la ontología original (postulados de **Éxito**).

Respecto de la relación entre las construcciones presentadas (local/kernel y global/cluster) y las propiedades esperadas que expresan los postulados, hemos establecido claramente la relación entre ambos aportes mediante distintos teoremas de representación (Teoremas 5.1 y 6.1). El aporte de tales teoremas es doble:

- por un lado, éstos teoremas muestran claramente como las construcciones propuestas satisfacen las propiedades esperadas para cada tipo de operador,
- y por el otro lado, los mismos muestran que cada operador de consolidación que sea definido de forma tal que satisfaga las propiedades en los teoremas se corresponderá necesariamente con alguna de las construcciones propuestas en este trabajo, es decir, que dada la misma ontología original obtendrá como resultado la misma ontología consolidada que se obtiene utilizando las construcciones presentadas en las Definiciones 5.6 y 6.6.

Apéndices

Apéndice A

Demostraciones

Por razones de legibilidad, a través de los diversos capítulos hemos presentado los diferentes resultados obtenidos en la presente línea de investigación (proposiciones, lemas, teoremas) omitiendo las demostraciones de los mismos. A continuación recapitulamos tales resultados, incluyendo además las pruebas de los mismos.

Proposición 3.1 [página 68] *Sea D un conjunto de átomos, $\Sigma_{_T}$ un conjunto de TGDs, $\Sigma_{_E}$ un conjunto de EGD separables y $\Sigma_{_{NC}}$ un conjunto de NCs. Sea $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ con $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$. Si KB es incoherente y existe $A \subseteq D$ tal que A es relevante a algún conjunto insatisfacible $U \subseteq \Sigma_{_T}$ entonces $KB = (D, \Sigma)$ es inconsistente.*

Prueba Sea $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ incoherente donde $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$. Considere un $U \subseteq \Sigma_{_T}$ tal que U es un conjunto insatisfacible de TGDs respecto de $\Sigma_{_{NC}} \cup \Sigma_{_E}$, y $A \subseteq D$ un conjunto de átomos relevante a U.

De la definición de satisfacibilidad de un conjunto de TGDs se sigue que si U es insatisfacible entonces no existe un conjunto relevante de átomos A' que haga que $\text{mods}(A', U \cup \Sigma_{_E} \cup \Sigma_{_{NC}}) \neq \emptyset$, debido a que de lo contrario U es satisfacible.

Entonces, $\text{mods}(A, U \cup \Sigma_{_E} \cup \Sigma_{_{NC}}) = \emptyset$. Entonces, como $A \subseteq D$ y $U \subseteq \Sigma_{_T}$ tenemos que $\mathit{chase}(A, U) \subseteq \mathit{chase}(D, \Sigma_{_T})$, y por lo tanto cualquier NC o EGD que es violada en $\mathit{chase}(A, U)$ es también violada en $\mathit{chase}(D, \Sigma_{_T})$. Por lo tanto, $\text{mods}(D, \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}) = \emptyset$, *esto es, KB* es inconsistente. ∎

Proposición 5.1 [página 108] *Las implicaciones entre postulados es como sigue:*

- *Si* Υ *es un operador de consolidación de ontologías Datalog*$^{\pm}$ *que satisface* ***Inclusión*** *y* ***Éxito para Datos****, entonces* Υ *satisface* ***Consistencia****.*
- *Si* Υ *es un operador de consolidación de ontologías Datalog*$^{\pm}$ *que satisface* ***Inclusión*** *y* ***Éxito para Restricciones****, entonces* Υ *satisface* ***Coherencia****.*
- *Si* Υ *es un operador de consolidación de ontologías Datalog*$^{\pm}$ *que satisface* ***Éxito para Restricciones****,* ***Vacuidad para Restricciones****,* ***Optimalidad Local para Restricciones****,* ***Éxito para Datos****,* ***Vacuidad para Datos*** *y* ***Optimalidad Local para Datos*** *entonces* Υ *satisface* ***Congruencia****.*

Prueba

- Sea Υ un operador que satisface **Inclusión** y **Éxito para Datos**. Entonces, Υ satisface **Consistencia**.

  Sea $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ donde $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$. Supongamos *por absurdo* que Υ no satisface **Consistencia**, *esto es,* $\text{mods}(\Upsilon(KB)) = \emptyset$.

  Entonces, existe un $X \subset D'$ tal que $\text{mods}(X, \Sigma') = \emptyset$ y para todo $X' \subsetneq X$ se verifica que $\text{mods}(X', \Sigma') \neq \emptyset$, ya que siempre se puede encontrar un subconjunto inconsistente minimal a partir de un conjunto inconsistente [Han01]. Como $X \subset D'$, por **Inclusión** tenemos que $X \subset D$.

  Sin embargo, como X es tal que $\text{mods}(X, \Sigma') = \emptyset$ y para todo $X' \subsetneq X$ se verifica que $\text{mods}(X', \Sigma') \neq \emptyset$, entonces por **Éxito para Datos** tenemos que existe $\alpha \in X$ tal que $\alpha \notin D'$. Entonces se verifica que $X \nsubseteq D'$.

  Por lo tanto, tenemos que $X \subseteq D'$ y $X \nsubseteq D'$, un absurdo que surge de nuestra suposición inicial de que $\text{mods}(\Upsilon(KB)) = \emptyset$, y entonces $\text{mods}(\Upsilon(KB)) \neq \emptyset$, *esto es,* $\Upsilon(KB)$ es consistente.

- Sea Υ un operador que satisface **Inclusión** y **Éxito para Restricciones**. Entonces, Υ satisface **Coherencia**.

  Omitiremos la prueba de que si un operador satisface **Inclusión** y **Éxito para Restricciones** entonces satisface **Coherencia** debido a que es análoga a la prueba anterior reemplazando **Éxito para Restricciones** por **Éxito para Datos**.

- Sea Υ un operador que satisface **Éxito para Restricciones**, **Vacuidad para Restricciones**, **Optimalidad Local para Restricciones**, **Éxito para Datos**, **Vacuidad para Datos** y **Optimalidad Local para Datos**. Entonces, Υ satisface **Congruencia**.

  Para probar esto, tenemos que mostrar que dadas dos ontologías Datalog$^{\pm}$ $KB_1 = (D_1, \Sigma_1)$ y $KB_2 = (D_2, \Sigma_2)$ se verifica que si $KB_1 = KB_2$ entonces $\Sigma_1' = \Sigma_2'$ y $D_1' = D_2'$, y por lo tanto $\Upsilon(KB_1) = \Upsilon(KB_2)$.

  Comenzaremos el análisis con el caso de $\Sigma_{1_T} \subseteq KB_1$ y $\Sigma_{2_T} \subseteq KB_2$. Considere $\alpha \in \Sigma_{1_T}$. Tenemos que mostrar que si $\alpha \notin \Sigma_1'$ entonces $\alpha \notin \Sigma_2'$, y si $\alpha \in \Sigma_1'$ entonces $\alpha \in \Sigma_2'$.

  Sea $\alpha \in \Sigma_{1_T}$ tal $\alpha \notin \Sigma_1'$. Entonces, por **Optimalidad Local para Restricciones** se verifica que existe $X \subseteq \Sigma_T$ tal que $X = \mu \cup \alpha$ y $\alpha \notin \mu$ donde μ es un conjunto satisfacible de TGDs y X es un conjunto insatisfacible de TGDs con respecto a Σ_1; donde para todo $\beta \in \mu$ se verifica que $\alpha \lessdot \beta$.

  Como $KB_1 = KB_2$, entonces $\Sigma_1 = \Sigma_2$ y $D_1 = D_2$, y se verifica que $\alpha \in \Sigma_2$, $\mu \subset X \subseteq \Sigma_{2_T}$ y $\alpha \lessdot \beta$ para KB_2 también.

  Asumamos *por absurdo* que $\alpha \in \Sigma_2'$. Tenemos que X es insatisfacible y $\mu = X \setminus \{\alpha\}$ es satisfacible. Como μ es satisfacible, entonces todo $\mu' \subseteq \mu$ es satisfacible, y X es tal que todo subconjunto propio del mismo es satisfacible, ya que no puede exstir ν tal que $\mu \subsetneq \nu \subset X$ porque μ surge de remover un solo elemento de X. Como $\alpha \in X$ donde X es insatisfacible y todo subconjunto propio de X es satisfacible y $\alpha \in \Sigma_2'$, entonces por **Éxito para Restricciones** debe valer que algún $\beta \in X$ es tal que $\beta \notin \Sigma_2'$, y entonces por **Optimalidad Local para Restricciones** tenemos que vale que $\beta \lessdot \alpha$. Además, $\beta \in \mu$. Esto contradice nuestro hipótesis de que $\alpha \lessdot \beta$ para todo $\beta \in \mu$, ya que $\lessdot$ es una relación antisimétrica. Por lo tanto, llegamos a un absurdo que surge de nuestra suposición de que $\alpha \in \Sigma_2'$, y por lo tanto tenemos que si $\alpha \in \Sigma_{1_T}$ es tal que $\alpha \notin \Sigma_1'$ entonces se verifica que $\alpha \notin \Sigma_2'$ (1).

  Ahora, considere el caso donde $\alpha \in \Sigma_{1_T}$ es tal que $\alpha \in \Sigma_1'$. Podemos distinguir dos situaciones diferentes aquí: o bien α pertenece a algún conjunto insatisfacible minimal de TGDs, o bien no lo hace. Esto es,

- o bien $\alpha \notin X$ para todo $X \subset \Sigma_{1_T}$ tal que X es insatisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$, o sino
- $\alpha \in X$ para algún $X \subset \Sigma_{1_T}$ tal que X es insatisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y existe $\beta \in X$ tal que $\beta \notin \Sigma'_1$.

Es conveniente analizar los casos por separado.

- Sea $\alpha \notin X$ para todo $X \subset \Sigma_{1_T}$ tal que X es insatisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$. Como $KB_1 = KB_2$, entonces $\Sigma_1 = \Sigma_2$ y tenemos que $\alpha \notin X$ para todo $X \subset \Sigma_{2_T}$ tal que X es insatisfacible con respecto a $\Sigma_{2_E} \cup \Sigma_{2_{NC}} \subset \Sigma_2$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{2_E} \cup \Sigma_{2_{NC}} \subset \Sigma_2$. Entonces, por **Vacuidad para Restricciones** tenemos que $\alpha \in D'_2$ (a).
- Ahora, consideremos el caso alternativo. Sea $\alpha \in X$ para algún $X \subset \Sigma_{1_T}$ tal que X es insatisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y existe $\beta \in X$ tal que $\beta \notin \Sigma'_1$.

  Como $\beta \in X$ y $\beta \notin \Upsilon(KB_1)$ entonces por **Optimalidad Local para Restricciones** tenemos que $\beta \lessdot \alpha$. Como $KB_1 = KB_2$ entonces se verifica que $\beta \lessdot \alpha$ para KB_2 también.

  Ahora, asumamos *por absurdo* que $\alpha \notin \Upsilon(KB_2)$. Como $KB_1 = KB_2$, entonces $\Sigma_1 = \Sigma_2$ y tenemos que $X \subset \Sigma_{2_T}$ es tal que X es insatisfacible con respecto a $\Sigma_{2_E} \cup \Sigma_{2_{NC}} \subset \Sigma_2$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{2_E} \cup \Sigma_{2_{NC}} \subset \Sigma_2$. Entonces, por **Optimalidad Local para Restricciones** se verifica que $\alpha \lessdot \beta$.

  Entonces, tenemos que $\beta \lessdot \alpha$ y $\alpha \lessdot \beta$, lo que solamente puede ocurrir si $\alpha = \beta$ ya que $\lessdot$ es una relación antisimétrica. Sin embargo, $\alpha = \beta$ es un absurdo (ya que por hipótesis $\alpha \in \Sigma'_1$ y $\beta \notin \Sigma'_1$), la cual proviene de nuestra suposición inicial de que $\alpha \notin \Upsilon(KB_2)$, y se verifica que si $\alpha \in X$ para algún $X \subset \Sigma_{1_T}$ tal que X es insatisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ entonces $\alpha \in \Upsilon(KB_2)$ (b).

De (a) y (b) se sigue que si $\alpha \in \Upsilon(KB_1)$ entonces $\alpha \in \Upsilon(KB_2)$ (2). Ahora, de (1), (2) y el hecho de que $\Sigma_{1_E}, \Sigma_{1_{NC}}, \Sigma_{2_E}$ y $\Sigma_{2_{NC}}$ se mantiene inalterados en el proceso de consolidación se verifica que $\Sigma'_1 = \Sigma'_2$.

Omitiremos la prueba de que si $KB_1 = KB_2$ entonces $D'_1 = D'_2$ usando **Éxito para Datos**, **Vacuidad para Datos** y **Optimidad Local para Datos** debido a que es análoga a la prueba recién mostrada de que si $KB_1 = KB_2$ entonces $\Sigma'_1 = \Sigma'_2$ usando **Éxito para Restricciones**, **Vacuidad para Restricciones** y **Optimidad Local para Restricciones**.

Finalmente, tenemos que si Υ es un operador que satisface **Éxito para Restricciones**, **Vacuidad para Restricciones**, **Optimidad Local para Restricciones**, **Éxito para Datos**, **Vacuidad para Datos** y **Optimidad Local para Datos** y $KB_1 = KB_2$, entonces se verifica que $\Sigma'_1 = \Sigma'_2$ y $D'_1 = D'_2$, y por lo tanto $\Upsilon(KB_1) = \Upsilon(KB_2)$. Esto es, si Υ es un operador que satisface **Éxito para Restricciones**, **Vacuidad para Restricciones**, **Optimidad Local para Restricciones**, **Éxito para Datos**, **Vacuidad para Datos** y **Optimidad Local para Datos** entonces Υ satisface **Congruencia**.

∎

Lema 5.2 [página 111] *Sean* $KB_1 = (D_1, \Sigma_1)$ *donde* $\Sigma_1 = \Sigma_{1_T} \cup \Sigma_{1_E} \cup \Sigma_{1_{NC}}$, *y* $KB_2 = (\emptyset, \Sigma_2)$ *donde* $\Sigma_2 = \Sigma_{2_T} \cup \Sigma_{2_E} \cup \Sigma_{2_{NC}}$ *dos ontologías Datalog*$^{\pm}$ *tales que* $\Sigma_1 = \Sigma_2$. *Entonces,* $\perp\!\!\!\perp_{(\Sigma_1, KB_1)} = \perp\!\!\!\perp_{(\Sigma_2, KB_2)}$.

Prueba Sea $KB_1 = (D_1, \Sigma_1)$ donde $\Sigma_1 = \Sigma_{1_T} \cup \Sigma_{1_E} \cup \Sigma_{1_{NC}}$, y $KB_2 = (\emptyset, \Sigma_2)$ donde $\Sigma_2 = \Sigma_{2_T} \cup \Sigma_{2_E} \cup \Sigma_{2_{NC}}$ dos ontologías Datalog$^{\pm}$ tales que $\Sigma_1 = \Sigma_2$, $\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ el conjunto de kernels de dependencias para KB_1, y $\perp\!\!\!\perp_{(\Sigma_2, KB_2)}$ el conjunto de dependencias para KB_2, respectivamente.

Considere cualquier $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$. Entonces, por Definición 5.1 tenemos que $X \subset \Sigma_{1_T}$ es un conjunto insatisfacible de dependencias con respecto a *w.r.t.* $\Sigma_{1_E} \cup \Sigma_{1_{NC}}$ y todo $X' \subsetneq X$ es un conjunto satisfacible de dependencias con respecto a *w.r.t.* $\Sigma_{1_E} \cup \Sigma_{1_{NC}}$. Como $\Sigma_1 = \Sigma_2$, entonces $\Sigma_{1_T} = \Sigma_{2_T}$, $\Sigma_{1_E} = \Sigma_{2_E}$ y $\Sigma_{1_{NC}} = \Sigma_{2_{NC}}$. Por lo tanto, se verifica que $X \subset \Sigma_{2_T}$ es un conjunto insatisfacible de dependencias con respecto a $\Sigma_{2_E} \cup \Sigma_{2_{NC}}$ y todo $X' \subsetneq X$ es un conjunto satisfacible de dependencias con respecto a *w.r.t.* $\Sigma_{2_E} \cup \Sigma_{2_{NC}}$.

Entonces, por Definición 5.1 tenemos que $X \in \perp\!\!\!\perp_{(\Sigma_2, KB_2)}$, y como esto vale para cualquier kernel de dependencias arbitrario en $\perp\!\!\!\perp_{(\Sigma, KB_1)}$ tenemos que $\perp\!\!\!\perp_{(\Sigma_1, KB_1)} = \perp\!\!\!\perp_{(\Sigma_2, KB_2)}$.

∎

Lema 5.3 [página 112] *Sean* $KB_1 = (D_1, \Sigma_1)$ *y* $KB_2 = (D_2, \Sigma_2)$ *dos ontologías Datalog*$^{\pm}$ *tales que* $KB_1 = KB_2$. *Entonces,* $\perp\!\!\!\perp_{(D_1,KB_1)} = \perp\!\!\!\perp_{(D_2,KB_2)}$ *y* $\perp\!\!\!\perp_{(\Sigma_1,KB_1)} = \perp\!\!\!\perp_{(\Sigma_2,KB_2)}$.

Prueba Sean $KB_1 = (D_1, \Sigma_1)$ donde $\Sigma_1 = \Sigma_{1_T} \cup \Sigma_{1_E} \cup \Sigma_{1_{NC}}$, y $KB_2 = (D_2, \Sigma_2)$ donde $\Sigma_2 = \Sigma_{2_T} \cup \Sigma_{2_E} \cup \Sigma_{2_{NC}}$ dos ontologías Datalog$^{\pm}$ tales que $KB_1 = KB_2$, $\perp\!\!\!\perp_{(\Sigma_1,KB_1)}$ y $\perp\!\!\!\perp_{(D_1,KB_1)}$ sean los kernels de dependencias y datos de KB_1 respectivamente, y $\perp\!\!\!\perp_{(D_2,KB_2)}$ y $\perp\!\!\!\perp_{(\Sigma_2,KB_2)}$ sean los kernels de dependencias y datos de KB_2, respectivamente.

Considere cualquier $X \in \perp\!\!\!\perp_{(\Sigma_1,KB_1)}$ arbitrario. Entonces, X es un conjunto minimal de TGDs insatisfacible con respecto de $\Sigma_{1_{NC}} \cup \Sigma_{1_E}$. Como $KB_1 = KB_2$, se verifica que $X \subset KB_2$, $\Sigma_{1_{NC}} = \Sigma_{2_{NC}}$, $\Sigma_{1_E} = \Sigma_{1_E}$ y X es un conjunto minimal de TGDs insatisfacible con respecto de $\Sigma_{2_{NC}} \cup \Sigma_{2_E}$. Además, no existe $X' \subsetneq X$ tal que X' es un conjunto minimal de TGDs insatisfacible con respecto de $\Sigma_{2_{NC}} \cup \Sigma_{2_E}$ porque, de lo contrario, contradecimos nuestra hipótesis de que $X \in \perp\!\!\!\perp_{(\Sigma_1,KB_1)}$, ya que $\Sigma_{1_{NC}} = \Sigma_{2_{NC}}$ y $\Sigma_{1_E} = \Sigma_{2_E}$. Por lo tanto, $X \in \perp\!\!\!\perp_{(\Sigma_2,KB_2)}$. Como esto vale para cualquier $X \in \perp\!\!\!\perp_{(\Sigma_1,KB_1)}$ arbitrario tenemos que $\perp\!\!\!\perp_{(\Sigma_1,KB_1)} = \perp\!\!\!\perp_{(\Sigma_2,KB_2)}$.

De forma análoga, considere cualquier $X \in \perp\!\!\!\perp_{(D_1,KB_1)}$ arbitrario. Entonces, X es un conjunto de átomos tal que $\text{mods}(X, \Sigma_1) = \emptyset$ y para cada $X' \subsetneq X$ se verifica que $\text{mods}(X', \Sigma_1) \neq \emptyset$. Como $KB_1 = KB_2$, $X \subset KB_2$ y $\Sigma_1 = \Sigma_2$. Entonces, $\text{mods}(X, \Sigma_2) = \emptyset$ y para cada $X' \subset X$ se verifica que $\text{mods}(X', \Sigma_2) \neq \emptyset$. Por lo tanto, $X \in \perp\!\!\!\perp_{(D_2,KB_2)}$. Como esto se verifica para cualquier $X \in \perp\!\!\!\perp_{(D_1,KB_1)}$ arbitrario tenemos que $\perp\!\!\!\perp_{(D_1,KB_1)} = \perp\!\!\!\perp_{(D_2,KB_2)}$. ∎

Proposición 5.4 [página 115] *Sea* $\mathcal{R}$ *un esquema relacional,* $KB = (D, \Sigma)$ *una ontología Datalog*$^{\pm}$ *sobre* $\mathcal{R}$ *donde* $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$, *y sea* $\perp\!\!\!\perp_{(\Sigma,KB)}$ *el conjunto de kernels de dependencias para* KB. *Sea* δ *una función de incisión general. Si* $\alpha \in D$ *es relevante a algún* $X \in \perp\!\!\!\perp_{(\Sigma,KB)}$ *entonces* $\alpha \in \delta(KB)$.

Prueba Sea $\mathcal{R}$ un esquema relacional, $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ sobre $\mathcal{R}$ donde $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$, y sea $\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de kernels de dependencias para KB. Sea δ una función de incisión general.

Considere $\alpha \in D$ y $X \in \perp\!\!\!\perp_{(\Sigma,KB)}$ tales que α es relevante a X. De la Definición 5.1 tenemos que X es insatisfacible con respecto a $N \subseteq \Sigma_E \cup \Sigma_{NC}$ y entonces, por Definición 3.4, y el hecho de que α es relevante a X tenemos que $\text{mods}(\{\alpha\}, X \cup N) = \emptyset$

(1). Además, como $\{\alpha\}$ es un conjunto unitario entonces el único $A \subsetneq \{\alpha\}$ es $A = \emptyset$, y claramente $\text{mods}(\emptyset, X \cup N) \neq \emptyset$ (2). Luego, de (1), (2) y de la Definición 5.2 surge que $\{\alpha\} \in \perp\!\!\!\perp_{(D,KB)}$.

Considere la incisión sobre $\{\alpha\}$. De la Definición 5.3 se sigue que $\delta(KB) \cap \{\alpha\} \neq \emptyset$. Entonces, tenemos que $\delta(KB) \cap \{\alpha\} = \alpha$ y, por lo tanto, $\alpha \in \delta(KB)$.

∎

Corolario 5.5 [página 116] *Sea $\mathcal{R}$ un esquema relacional, $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ sobre $\mathcal{R}$ donde $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$, y sea $\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de kernels de dependencias para KB. Sea δ una función de incisión general. Si para todo $\alpha \in D$ se verifica que α es relevante a algún $X \in \perp\!\!\!\perp_{(\Sigma,KB)}$ entonces $D \subseteq \delta(KB)$.*

Prueba Sea $\mathcal{R}$ un esquema relacional, $KB = (D, \Sigma)$ una ontología Datalog$^{\pm}$ sobre $\mathcal{R}$ donde $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$, y sea $\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de kernels de dependencias para KB. Sea δ una función de incisión general.

Considere cualquier $\alpha \in D$. Como α es relevante a algún $X \in \perp\!\!\!\perp_{(\Sigma,KB)}$, entonces por Proposición 5.4 se verifica que $\alpha \in \delta(KB)$. Por lo tanto, como esto vale para cualquier $\alpha \in D$ arbitrario tenemos que $D \subseteq \delta(KB)$. ∎

Teorema 5.1 [página 119] *El operador $\Upsilon_{\rho,\varrho}$ es un* operador de consolidación de ontologías Datalog$^{\pm}$ basado en Kernel Contraction *para una ontología Datalog$^{\pm}$ KB si y sólo si satisface* ***Inclusión***, ***Éxito para Restricciones***, ***Éxito para Datos***, ***Vacuidad para Restricciones***, ***Vacuidad para Datos***, ***Optimalidad Local para Restricciones*** *y* ***Optimalidad Local para Datos***.

Prueba Sean KB_1 y KB_2 dos ontologías de la forma $KB_1 = (D_1, \Sigma_1)$ y $KB_2 = (D_2, \Sigma_2)$, respectivamente, tales que $KB_1 = KB_2$.

⇒) Construcción a Postulados

Considere un operador $\Upsilon_{\rho,\varrho}$ definido como en la Definición 5.6. Tenemos que probar que $\Upsilon_{\rho,\varrho}$ satisface los postulados en el Teorema 5.1. Sean $\Upsilon_{\rho,\varrho}(KB_1) = (D_1', \Sigma_1')$ y $\Upsilon_{\rho,\varrho}(KB_2) = (D_2', \Sigma_2')$ dos ontologías Datalog$^{\pm}$ resultantes de la consolidación de KB_1 y KB_2 por medio de $\Upsilon_{\rho,\varrho}$, respectivamente.

Además, sea $KB_1^\star = (D_1, \Sigma_1 \setminus \rho(KB_1))$ y $KB_2^\star = (D_2, \Sigma_2 \setminus \rho(KB_2))$ la ontología Datalog$^\pm$ resultante de eliminar de KB_1 y KB_2, respectivamente, las TGD seleccionada por ρ. Finalmente, sean $\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ y $\perp\!\!\!\perp_{(D_1, KB_1^\star)}$ los conjuntos de kernels de dependencias y kernels de datos para KB_1 y $KB_1^\star$ respectivamente, $\perp\!\!\!\perp_{(\Sigma_2, KB_2)}$ y $\perp\!\!\!\perp_{(D_2, KB_2^\star)}$ los conjuntos de kernels de dependencias y kernels de datos para KB_2 y $KB_2^\star$ respectivamente, $\preccurlyeq_1$ un orden para las fórmulas en KB_1, y $\preccurlyeq_2$ un orden para las fórmulas en KB_2.

- **Inclusión**: $\Sigma_1' \subseteq \Sigma_1$ y $D_1' \subseteq D_1$.

  Por definición de $\Upsilon_{\rho,\varrho}(KB_1)$ tenemos que $D_1' = D_1 \setminus \varrho(KB_1^\star)$, y por lo tanto $D_1' \subseteq D_1$.

  De forma similar, por definición de $\Upsilon_{\rho,\varrho}(KB_1)$ tenemos que $\Sigma_1' = \Sigma_1 \setminus \rho(KB_1)$, y por lo tanto $\Sigma_1' \subseteq \Sigma_1$.

- **Éxito para Restricciones**: Para todo $X \subset \Sigma_{1_T}$ tal que X es insatisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ existe $\alpha \in X$ tal que $\alpha \notin \Sigma_1'$.

  De la definición de ρ tenemos que para todo $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ y $X \neq \emptyset$ se verifica que $(\rho(KB_1) \cap X) \neq \emptyset$. Entonces, existe algún $\alpha \in X$ tal que $\alpha \in \{\rho(KB_1) \cap X\}$, y por lo tanto $\alpha \notin \Sigma_1'$. Además, por Definición 5.1 tenemos que si $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ entonces $X \subset \Sigma_{1_T}$ tal que X es insatisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$. Por lo tanto, se verifica que para todo $X \subset \Sigma_{1_T}$ tal que X es insatisfacible con respecto ar $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ existe algún $\alpha \in X$ tal que $\alpha \notin \Sigma_1'$.

- **Éxito para Datos**: Prueba omitida debido a que es análoga a la prueba para **Exito para Restricciones**.

- **Vacuidad para Restricciones**: Si no existe $X \subset \Sigma_{1_T}$ tal que X es insatisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$, entonces $\alpha \in \Sigma_1'$.

  Sea α tal que no existe $X \subset \Sigma_{1_T}$ tal que X es insatisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$. Entonces, por Definición 5.1 se verifica que $\alpha \notin X$ para todo $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$. Por lo tanto, $\alpha \notin \bigcup(\perp\!\!\!\perp_{(\Sigma_1, KB_1)})$. Entonces, por definición de ρ tenemos que $\alpha \notin \rho(KB_1)$, y por Definición 5.6 se verifica que $\alpha \in \Sigma_1'$.

- **Vacuidad para Datos**: Prueba omitida debido a que es análoga a la prueba para **Vacuidad para Restricciones**.

- **(Optimalidad Local para Restricciones)**: Si $\alpha \in \Sigma_1$ y $\alpha \notin \Sigma_1'$, entonces existe $\mu \subseteq \Sigma_{1_T}$ tal que μ es satisfacible y $\alpha \cup \mu$ es insatisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subseteq \Sigma_1$ y para todo $\beta \in \mu$ se verifica que $\alpha \lessdot \beta$.

  Sea $\alpha \in \Sigma_1$ tal que $\alpha \notin \Sigma_1'$. Como $\alpha \notin \Sigma_1'$, entonces por definición de $\Upsilon_{\rho,\varrho}$ tenemos que $\alpha \in \rho(KB_1)$. Sea $\mu = X \setminus \{\alpha\}$. Como $\mu \subsetneq X$, por Definición 5.1 tenemos que $X = \mu \cup \alpha$ es insatisfacible y μ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}}$.

  Considere cualquier $\beta \in \mu$ arbitrario. Como $\mu \subset X$, entonces por Definición 5.4 tenemos que $\alpha \lessdot_1 \beta$.

- **Optimalidad Local para Datos**: Prueba omitida debido a que es análoga a la prueba para **Optimalidad Local para Restricciones**.

$\Leftarrow$) Postulados a Construcción

Para la segunda parte de la prueba, considere un operador $\Upsilon_{\rho,\varrho}$ que satisface todos los postulados en el Teorema 5.1. Sea $\rho_{(\Upsilon)_\Sigma}$ un función definida en base a $\Upsilon_{\rho,\varrho}$ como sigue:

$$\rho_{(\Upsilon)_\Sigma}(KB_1) = \{x \,|\, x \in X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)} \textit{and}\, x \notin \{\Sigma_1 \cap \Upsilon_{\rho,\varrho}(KB_1)\}\}$$

Sea $KB_1^\star = (D_1, \Sigma_1 \setminus \rho_{(\Upsilon)_\Sigma}(KB_1))$ la ontología resultante de eliminar de KB_1 las TGDs seleccionadas por $\rho_{(\Upsilon)_\Sigma}$. Entonces, sea $\varrho_{(\Upsilon)_D}$ otra función definida en base a $\Upsilon_{\rho,\varrho}$ como sigue:

$$\varrho_{(\Upsilon)_D}(KB_1^\star) = \{x \,|\, x \in X \in \perp\!\!\!\perp_{(D_1, KB_1^\star)} \textit{and}\, x \notin \{D_1 \cap \Upsilon_{\rho,\varrho}(KB_1)\}\}$$

En base a $\varrho_{(\Upsilon)_D}$ y $\rho_{(\Upsilon)_\Sigma}$ definimos un nuevo operador como sigue:

$$\Upsilon'_{\rho,\varrho}(KB_1) = (D_1 \setminus \varrho_{(\Upsilon)_D}(KB_1^\star), \Sigma_1 \setminus \rho_{(\Upsilon)_\Sigma}(KB_1))$$

Tenemos que mostrar que $\Upsilon'_{\rho,\varrho}$ es un *operador de consolidación de ontologías Datalog$^\pm$ basado en Kernel Contraction*. Para hacer esto, primero mostramos que $\varrho_{(\Upsilon)_D}$ es una *función de incisión en datos* bien definida y que $\rho_{(\Upsilon)_\Sigma}$ es una *función de incisión en restricciones* bien definida. Esto es, dada $\rho_{(\Upsilon)_\Sigma}$ tenemos que probar que:

- $\rho_{(\Upsilon)_\Sigma}$ esta bien definida, *esto es,* si $KB_1 = KB_2$, entonces $\rho_{(\Upsilon)_\Sigma}(KB_1) = \rho_{(\Upsilon)_\Sigma}(KB_2)$.

  Por definición de $\rho_{(\Upsilon)_\Sigma}$ tenemos que $\rho_{(\Upsilon)_\Sigma}(KB_1) = \{x \,|\, x \in X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)} \textit{and}\, x \notin \Sigma_1 \cap \Upsilon_{\rho,\varrho}(KB_1)\}$.

  Considere cualquier $x \in \rho_{(\Upsilon)_\Sigma}(KB_1)$. Como $KB_1 = KB_2$, entonces por Lema 5.3 tenemos que $\perp\!\!\!\perp_{(\Sigma_1, KB_1)} = \perp\!\!\!\perp_{(\Sigma_2, KB_2)}$. Como $x \in \rho_{(\Upsilon)_\Sigma}(KB_1)$, entonces $x \in X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$, y por lo tanto se verifica que $x \in X \in \perp\!\!\!\perp_{(\Sigma_2, KB_2)}(1)$.

  Ademas, como $x \in X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ entonces $x \in \Sigma_1$. Por lo tanto, como $x \notin \Sigma_1 \cap \Upsilon_{\rho,\varrho}(KB_1)$, entonces $x \notin \Upsilon_{\rho,\varrho}(KB_1)$. De la Proposición 5.1 se sigue que si $\Upsilon_{\rho,\varrho}$ satisface **Éxito para Restricciones**, **Éxito para Datos**, **Optimalidad Local para Restricciones**, y **Optimalidad Local para Datos** entonces éste satisface *Congruencia*. Como $KB_1 = KB_2$, por *Congruencia* tenemos que $\Upsilon_{\rho,\varrho}(KB_1) = \Upsilon_{\rho,\varrho}(KB_2)$, y entonces vale también que $x \notin \Upsilon_{\rho,\varrho}(KB_2)$. Por lo tanto, $x \notin \Sigma_2 \cap \Upsilon_{\rho,\varrho}(KB_2)(2)$.

  De (1) y (2) se sigue que para cualquier $x \in \rho_{(\Upsilon)_\Sigma}(KB_1)$ se verifica que $x \in \{y \,|\, y \in Y \in \perp\!\!\!\perp_{(\Sigma_2, KB_2)}$ y $y \notin \Sigma_2 \cap \Upsilon_{\rho,\varrho}(KB_2)\}$. Por definición de $\rho_{(\Upsilon)_\Sigma}$, éste último conjunto es $\rho_{(\Upsilon)_\Sigma}(KB_2)$, y por lo tanto tenemos que si $KB_1 = KB_2$, entonces $\rho_{(\Upsilon)_\Sigma}(KB_1) = \rho_{(\Upsilon)_\Sigma}(KB_2)$.

- $\rho_{(\Upsilon)_\Sigma}(KB_1) \subseteq \bigcup(\perp\!\!\!\perp_{(\Sigma_1, KB_1)})$.

  Esto es obtenido directamente de la definición de $\rho_{(\Upsilon)_\Sigma}$, ya que para todo $x \in \rho_{(\Upsilon)_\Sigma}(KB_1)$ se verifica que $x \in X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ debido a la primer condición de la definición.

- Para todo $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ tal que $X \neq \emptyset$ se verifica que $(X \bigcap \rho_{(\Upsilon)_\Sigma}(KB_1)) \neq \emptyset$.

  Supongamos *por absurdo* que existe algún $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ tal que $X \neq \emptyset$ y $(X \bigcap \rho_{(\Upsilon)_\Sigma}(KB_1)) = \emptyset$.

  Entonces, para todo $\alpha \in X$ se verifica que $\alpha \notin \rho_{(\Upsilon)_\Sigma}(KB_1)$, *esto es,* $\alpha \notin X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ o $\alpha \in \{\Sigma_1 \cap \Upsilon_{\rho,\varrho}(KB_1)\}$. Por hipótesis tenemos que $\alpha \in X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$, y entonces $\alpha \in \{\Sigma_1 \cap \Upsilon_{\rho,\varrho}(KB_1)\}$, y por extensión $\alpha \in \Upsilon_{\rho,\varrho}(KB_1)$.

  Como $X \subset \Sigma_1$, entonces $\alpha \in \Sigma_1$. Como $\alpha \in \Upsilon_{\rho,\varrho}(KB_1)$ y $\alpha \in \Sigma_1$, entonces por **Éxito para Restricciones** tenemos que existe algún $\beta \in X$ tal que $\beta \notin \Upsilon_{\rho,\varrho}(KB_1)$. Entonces, $\beta \notin \{\Sigma_1 \cap \Upsilon_{\rho,\varrho}(KB_1)\}$. Por lo tanto, $\beta \in \rho_{(\Upsilon)_\Sigma}(KB_1)$; y $\beta \in (X \bigcap \rho_{(\Upsilon)_\Sigma}(KB_1))$. Entonces, $(X \bigcap \rho_{(\Upsilon)_\Sigma}(KB_1)) \neq \emptyset$.

Por lo tanto tenemos que $(X \bigcap \rho_{(\Upsilon)_\Sigma}(KB_1)) = \emptyset$ y $(X \bigcap \rho_{(\Upsilon)_\Sigma}(KB_1)) \neq \emptyset$, un absurdo que surge de nuestra suposición inicial de que existe algún $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ tal que $X \neq \emptyset$ y $(X \bigcap \rho_{(\Upsilon)_\Sigma}(KB_1)) = \emptyset$, y se verifica que para todo $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ tal que $X \neq \emptyset$ se verifica que $(X \bigcap \rho_{(\Upsilon)_\Sigma}(KB_1)) \neq \emptyset$.

- Para todo $\sigma \in (\rho_{(\Upsilon)_\Sigma}(KB_1))$ existe $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ tal que para cualquier $\sigma' \in X$ se verifica que $\sigma \preccurlyeq \sigma'$.

  Sea $\sigma \in (\rho_{(\Upsilon)_\Sigma}(KB_1))$. Por definición de $\rho_{(\Upsilon)_\Sigma}$ tenemos que $\rho_{(\Upsilon)_\Sigma}(KB_1) = \{x \,|\, x \in X \text{ for some } X \in \text{ for some } X \perp\!\!\!\perp_{(\Sigma_1, KB_1)} \text{ and } x \notin \Sigma_1 \cap \Upsilon_{\rho,\varrho}(KB_1)\}$. Entonces, se verifica que $\sigma \in \Sigma_{1_T}$ y $\sigma \notin \Sigma'_1$. Por la tanto, por **Optimalidad Local para Restricciones** tenemos que existe algún $\mu \subset \Sigma_{1_T}$ tal que μ es satisfacible y $\sigma \cup \mu$ es insatisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}}$. Por Definición 5.1 esto significa que para $X = \sigma \cup \mu$ se verifica que $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$.

  Queda mostrar que para toda $\sigma' \in X$ se verifica que $\sigma \preccurlyeq_1 \sigma'$. Considere algún $\sigma' \in X$ arbitrario. Si $\sigma' = \sigma$, entonces $\sigma \preccurlyeq_1 \sigma'$ de forma trivial, ya que $\preccurlyeq_1$ es reflexiva y transitiva. Si $\sigma' \neq \sigma$, entonces $\sigma' \in \mu$. Entonces, por **Optimalidad Local para Restricciones** tenemos que $\sigma \preccurlyeq_1 \sigma'$.

Omitiremos la prueba de que $\varrho_{(\Upsilon)_D}$ es una *función de incisión en datos* bien definida usando **Éxito para Datos**, **Vacuidad para Datos** y **Optimalidad Local para Datos** ya que la misma es análoga a la prueba de que $\rho_{(\Upsilon)_\Sigma}$ es una *función de incisión en restricciones* bien definida usando **Éxito para Restricciones**, **Vacuidad para Restricciones** y **Optimalidad Local para Restricciones**.

Para finalizar esta segunda parte de la prueba, ahora que hemos demostrado que $\varrho_{(\Upsilon)_D}$ es una *función de incisión en datos* bien definida y que $\rho_{(\Upsilon)_\Sigma}$ es una *función de incisión en restricciones* bien definida, tenemos que probar que $\Upsilon'_{\rho,\varrho}$ coincide con $\Upsilon_{\rho,\varrho}$. De **Inclusión** se sigue que $D'_1 \subseteq D_1$ y $\Sigma'_1 \subseteq \Sigma_1$ (1). Ademas, de nuestra definición de $\varrho_{(\Upsilon)_D}$ surge que $\varrho_{(\Upsilon)_D}(KB^\star_1) = D_1 \setminus D'_1$, y de nuestra definición de $\rho_{(\Upsilon)_\Sigma}$ surge que $\rho_{(\Upsilon)_\Sigma}(KB_1) = \Sigma_1 \setminus \Sigma'_1$ (2).

Entonces, por (1) y (2) tenemos que $D'_1 = D_1 \setminus \varrho_{(\Upsilon)_D}(KB^\star_1)$ y $\Sigma'_1 = \Sigma_1 \setminus \rho_{(\Upsilon)_\Sigma}(KB_1)$. Por lo tanto,

$$\Upsilon_{\rho,\varrho} = (D_1 \setminus \varrho_{(\Upsilon)_D}(KB^\star_1), \Sigma_1 \setminus \rho_{(\Upsilon)_\Sigma}(KB_1))$$

y entonces $\Upsilon'_{\rho,\varrho}$ coincide con $\Upsilon_{\rho,\varrho}$. ∎

Corolario 5.6 [página 119] *El operador $\Upsilon_{\rho,\varrho}$ satisface **Coherencia** y **Consistencia**.*

Prueba Se sigue de la Proposición 5.1 que si un operador satisface **Inclusión** y **Éxito para Datos** entonces satisface **Consistencia**, y si satisface **Inclusión** y **Éxito para Restricciones** entonces satisface **Coherencia**. ∎

Proposición 6.1 [página 131] *Sea KB una ontología Datalog$^{\pm}$, $\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de kernels de dependencias para KB y $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de clusters de dependencias obtenido en base a $\perp\!\!\!\perp_{(\Sigma,KB)}$, $\perp\!\!\!\perp_{(D,KB)}$ el conjunto de kernels de datos para KB y $\perp\!\!\!\perp_{(D,KB)}$ el conjunto de clusters de datos para KB obtenido en base a $\perp\!\!\!\perp_{(D,KB)}$.*

Entonces, $Y \in \perp\!\!\!\perp_{(\Sigma,KB)}$ es tal que $Y \subseteq X$ para algún $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ si y solo si $Y \nsubseteq X'$ para todo $X' \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ tal que $X \neq X'$. Análogamente, $Y \in \perp\!\!\!\perp_{(D,KB)}$ es tal que $Y \subseteq X$ para algún $X \in \perp\!\!\!\perp\!\!\!\perp_{(D,KB)}$ si y solo si $Y \nsubseteq X'$ para todo $X' \in \perp\!\!\!\perp\!\!\!\perp_{(D,KB)}$ tal que $X \neq X'$.

Prueba Sea KB una ontología Datalog$^{\pm}$, $\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de kernels de dependencias para KB y $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ el conjunto de clusters de dependencias obtenido en base a $\perp\!\!\!\perp_{(\Sigma,KB)}$, $\perp\!\!\!\perp_{(D,KB)}$ el conjunto de kernels de datos para KB y $\perp\!\!\!\perp_{(D,KB)}$ el conjunto de clusters de datos para KB obtenido en base a $\perp\!\!\!\perp_{(D,KB)}$.

Nos enfocaremos en el caso de los clusters de dependencias, omitiendo las pruebas para el caso de clusters de datos ya que las mismas son análogas. Considere cualquier $Y \in \perp\!\!\!\perp_{(\Sigma,KB)}$ arbitrario.

⇒) Primero mostramos que si un kernel es parte de un cluster entonces no es parte de ningún otro cluster, *esto es,* si $Y \subseteq X$ para algún $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ entonces $Y \nsubseteq X'$ para todo $X' \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB)}$ tal que $X \neq X'$.

Esto se obtiene directamente de la definiciones de clusters: tenemos que $X = \bigcup_{Y \in [\kappa]} Y$ donde $[\kappa]$ es una clase de equivalencia en la relación de equivalencia θ^* obtenida sobre $\perp\!\!\!\perp_{(\Sigma,KB)}$. Entonces, claramente para Y tenemos que si $Y \subseteq X$ entonces $Y \in [\kappa]$. Por lo tanto, como por definición dos clases de equivalencia son o iguales o disjuntas entonces vale que $Y \notin [\kappa']$ para toda $[\kappa']$. Sea $X' = \bigcup_{Y' \in [\kappa']} Y'$. Entonces se verifica que $X \neq X'$ y

que $Y \nsubseteq X'$. Como esto vale para cualquier clase de equivalencia $[\kappa']$ arbitraria entonces vale que si $Y \subseteq X$ para algún $X \in \text{⫲}_{(\Sigma,KB)}$ entonces $Y \nsubseteq X'$ para todo $X' \in \text{⫲}_{(\Sigma,KB)}$ tal que $X \neq X'$.

$\Leftarrow$) Ahora mostramos que no existe kernel que no pertenezca a un cluster, *esto es,* si $Y \nsubseteq X'$ para todo $X \in \text{⫲}_{(\Sigma,KB)}$ tal que $X \neq X'$ entonces $Y \subseteq X$ para $X \in \text{⫲}_{(\Sigma,KB)}$. Nuevamente, esto surge del uso de clases de equivalencia en las Definiciones 6.2 y 6.3. Si $Y \nsubseteq X'$ para todo $X' \in \text{⫲}_{(\Sigma,KB)}$ tal que $X \neq X'$, entonces vale que $Y \notin [\kappa']$ para toda $[\kappa'] \neq [\kappa]$. Entonces, como las clases de equivalencia forman una partición, debe valer que $Y \in [\kappa]$. Por lo tanto, como $X = \bigcup_{Y \in [\kappa] Y}$ tenemos que si $Y \nsubseteq X'$ para todo $X' \in \text{⫲}_{(\Sigma,KB)}$ tal que $X \neq X'$ entonces $Y \subseteq X$. ∎

Corolario 6.2 [página 131] *Sea KB una ontología Datalog$^{\pm}$, $\text{⫴}_{(\Sigma,KB)}$ el conjunto de kernels de dependencias para KB y $\text{⫲}_{(\Sigma,KB)}$ el conjunto de clusters de dependencias obtenido en base a $\text{⫴}_{(\Sigma,KB)}$, $\text{⫴}_{(D,KB)}$ el conjunto de kernels de datos para KB y $\text{⫴}_{(D,KB)}$ el conjunto de clusters de datos para KB obtenido en base a $\text{⫴}_{(D,KB)}$.*

Sea $\alpha \in Y$ para algún $Y \in \text{⫴}_{(\Sigma,KB)}$ y $\beta \in Y'$ para algún $Y' \in \text{⫴}_{(D,KB)}$. Entonces, $\alpha \in X$ para algún $X \in \text{⫲}_{(\Sigma,KB)}$ si y solo si $\alpha \notin X'$ para todo $X' \in \text{⫲}_{(\Sigma,KB)}$ tal que $X \neq X'$. Análogamente, $\beta \in Y'$ es tal que $\beta \in X$ para algún $X \in \text{⫲}_{(D,KB)}$ si y solo si $\beta \notin X'$ para todo $X' \in \text{⫲}_{(D,KB)}$ tal que $X \neq X'$.

Prueba Sea KB una ontología Datalog$^{\pm}$, $\text{⫴}_{(\Sigma,KB)}$ el conjunto de kernels de dependencias para KB y $\text{⫲}_{(\Sigma,KB)}$ el conjunto de clusters de dependencias obtenido en base a $\text{⫴}_{(\Sigma,KB)}$, $\text{⫴}_{(D,KB)}$ el conjunto de kernels de datos para KB y $\text{⫴}_{(D,KB)}$ el conjunto de clusters de datos para KB obtenido en base a $\text{⫴}_{(D,KB)}$. Finalmente, sea $\alpha \in Y$ para algún $Y \in \text{⫴}_{(\Sigma,KB)}$ y $\beta \in Y'$ para algún $Y' \in \text{⫴}_{(D,KB)}$.

Considere $\alpha \in Y$ para algún $Y \in \text{⫴}_{(\Sigma,KB)}$. Por Proposición 6.1 tenemos que $Y \subseteq X$ para algún $X \in \text{⫲}_{(\Sigma,KB)}$ si y solo si $Y \nsubseteq X'$ para todo $X' \in \text{⫲}_{(\Sigma,KB)}$ tal que $X \neq X'$. Por lo tanto, tenemos que $\alpha \in X$ para algún $X \in \text{⫲}_{(\Sigma,KB)}$ si y solo si $\alpha \notin X'$ para todo $X' \in \text{⫲}_{(\Sigma,KB)}$ tal que $X \neq X'$.

Análogamente, podemos probar que $\beta \in Y'$ es tal que $\beta \in X$ para algún $X \in \text{⫲}_{(D,KB)}$ si y solo si $\beta \notin X'$ para todo $X' \in \text{⫲}_{(D,KB)}$ tal que $X \neq X'$. ∎

Lema 6.3 [página 132] *Sean KB_1 y KB_2 dos ontologías Datalog$^{\pm}$ tales que $KB_1 = KB_2$. Entonces, $\perp\!\!\!\perp\!\!\!\perp_{(D,KB_1)} = \perp\!\!\!\perp\!\!\!\perp_{(D,KB_2)}$ y $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB_1)} = \perp\!\!\!\perp\!\!\!\perp_{(\Sigma,KB_2)}$.*

Prueba

Sean $KB_1 = \{D_1, \Sigma_1\}$ donde $\Sigma_1 = \Sigma_{1_T} \cup \Sigma_{1_E} \cup \Sigma_{1_T}$, y $KB_2 = \{D_2, \Sigma_2\}$ donde $\Sigma_2 = \Sigma_{2_T} \cup \Sigma_{2_E} \cup \Sigma_{2_T}$ dos ontologías Datalog$^{\pm}$ tales que $KB_1 = KB_2$, $\perp\!\!\!\perp_{(\Sigma_1,KB_1)}$ y $\perp\!\!\!\perp_{(D_1,KB_1)}$ los kernels de dependencias y datos de KB_1 respectivamente, y $\perp\!\!\!\perp_{(\Sigma_2,KB_2)}$ y $\perp\!\!\!\perp_{(D_2,KB_2)}$ los kernels de dependencias y datos de KB_2, respectivamente. Ademas, sea $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1,KB_1)}$ y $\perp\!\!\!\perp\!\!\!\perp_{(D_1,KB_1)}$ los clusters de dependencias y datos de KB_1 respectivamente, y $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma_2,KB_2)}$ y $\perp\!\!\!\perp\!\!\!\perp_{(D_2,KB_2)}$ los clusters de dependencias y datos de KB_2, respectivamente.

La prueba se sigue directamente del Lema 5.3 y la forma en que los clusters son obtenidos a partir de los kernels. Por Lema 5.3 tenemos que como $KB_1 = KB_2$ entonces $\perp\!\!\!\perp_{(\Sigma_1,KB_1)} = \perp\!\!\!\perp_{(\Sigma_2,KB_2)}$ y $\perp\!\!\!\perp_{(D_1,KB_1)} = \perp\!\!\!\perp_{(D_2,KB_2)}$.

Considere cualquier $X, Y \in \perp\!\!\!\perp_{(\Sigma_1,KB_1)}$ arbitrarios tales que $X\theta Y$. Como $\perp\!\!\!\perp_{(\Sigma_1,KB_1)} = \perp\!\!\!\perp_{(\Sigma_2,KB_2)}$, entonces $X, Y \in \perp\!\!\!\perp_{(\Sigma_2,KB_2)}$. Por lo tanto, $\theta^*_{\perp\!\!\!\perp_{(\Sigma_1,KB_1)}}$ es equivalente a $\theta^*_{\perp\!\!\!\perp_{(\Sigma_2,KB_2)}}$, y entonces $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1,KB_1)} = \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_2,KB_2)}$.

De forma análoga, considere cualquier $X', Y' \in \perp\!\!\!\perp_{(D_1,KB_1)}$ arbitrarios tales que $X'\theta Y'$. Como $\perp\!\!\!\perp_{(D_1,KB_1)} = \perp\!\!\!\perp_{(D_2,KB_2)}$, entonces $X', Y' \in \perp\!\!\!\perp_{(D_2,KB_2)}$. Por lo tanto, $\theta^*_{\perp\!\!\!\perp_{(D_1,KB_1)}}$ es equivalente a $\theta^*_{\perp\!\!\!\perp_{(D_2,KB_2)}}$, y entonces $\perp\!\!\!\perp\!\!\!\perp_{(D_1,KB_1)} = \perp\!\!\!\perp\!\!\!\perp_{(D_2,KB_2)}$.

∎

Teorema 6.1 [página 136] *El operador $\Psi_{\phi,\varphi}$ es un* operador de consolidación de ontologías Datalog$^{\pm}$ basado en Cluster Contraction *para una ontología Datalog$^{\pm}$ KB si y sólo si satisface* ***Inclusión***, ***Éxito para Restricciones***, ***Éxito para Datos***, ***Vacuidad para Restricciones***, ***Vacuidad para Datos*** *y* ***Mínima Pérdida de Información***. **Prueba**

Sean KB_1 y KB_2 dos ontologías de la forma $KB_1 = (D_1, \Sigma_1)$ y $KB_2 = (D_2, \Sigma_2)$, respectivamente, tales que $KB_1 = KB_2$.

⇒) Construcción a postulados

Considere un operador $\Psi_{\phi,\varphi}$ definido como en la Definición 6.6. Tenemos que probar que $\Psi_{\phi,\varphi}$ satisface los postulados en el Teorema 6.1. Sean $\Psi_{\phi,\varphi}(KB_1) = (D'_1, \Sigma'_1)$ y

$\Psi_{\phi,\varphi}(KB_2) = (D_2', \Sigma_2')$ dos ontologías Datalog$^{\pm}$ resultantes de la consolidación de KB_1 y KB_2 por medio de $\Psi_{\phi,\varphi}$, respectivamente.

Además, sea $KB_1^{\star} = (D_1, \Sigma_1 \setminus \phi(KB_1))$ y $KB_2^{\star} = (D_2, \Sigma_2 \setminus \phi(KB_2))$ la ontología Datalog$^{\pm}$ resultante de eliminar de KB_1 y KB_2, respectivamente, las TGD seleccionada por ϕ. Sean $\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ y $\perp\!\!\!\perp_{(D_1, KB_1^{\star})}$ los conjuntos de kernels de dependencias y kernels de datos para KB_1 y $KB_1^{\star}$ respectivamente, $\perp\!\!\!\perp_{(\Sigma_2, KB_2)}$ y $\perp\!\!\!\perp_{(D_2, KB_2^{\star})}$ los conjuntos de kernels de dependencias y kernels de datos para KB_2 y $KB_2^{\star}$ respectivamente. Finalmente, sean $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ y $\perp\!\!\!\perp\!\!\!\perp_{(D_1, KB_1^{\star})}$ los conjuntos de clusters de dependencias y clusters de datos para KB_1 y $KB_1^{\star}$ respectivamente, $\perp\!\!\!\perp\!\!\!\perp_{(\Sigma_2, KB_2)}$ y $\perp\!\!\!\perp\!\!\!\perp_{(D_2, KB_2^{\star})}$ los conjuntos de clusters de dependencias y clusters de datos para KB_2 y $KB_2^{\star}$ respectivamente.

- **Inclusión**: $\Sigma_1' \subseteq \Sigma_1$ y $D_1' \subseteq D_1$.

  Por definición de $\Psi_{\phi,\varphi}(KB_1)$ tenemos que $D_1' = D_1 \setminus \varphi(KB_1^{\star})$, y por lo tanto $D_1' \subseteq D_1$.

  De forma similar, por definición de $\Psi_{\phi,\varphi}(KB_1)$ tenemos que $\Sigma_1' = \Sigma_1 \setminus \phi(KB_1)$, y por lo tanto $\Sigma_1' \subseteq \Sigma_1$.

- **Éxito para Restricciones**: Para todo $X \subset \Sigma_{1_T}$ tal que X es insatisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ existe $\alpha \in X$ tal que $\alpha \notin \Sigma_1'$.

  Por definición de ϕ, para todo $Y \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ y $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ donde $X \subseteq Y$ se verifica que $(\phi(KB_1) \cap\ X) \neq \emptyset$. Entonces, existe algún $\alpha \in X$ tal que $\alpha \in \{\phi(KB_1) \cap\ X\}$, y por lo tanto $\alpha \notin \Sigma_1'$.

  Por Definición 5.1 tenemos que si $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ entonces X es insatisfacible respecto de $\Sigma_{1_{NC}} \cup \Sigma_{1_E} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible respecto de $\Sigma_{1_{NC}} \cup \Sigma_{1_E} \subset \Sigma_1$. Por lo tanto, se verifica que para todo $X \subset \Sigma_{1_T}$ tal que X es insatisfacible con respecto ar $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ y todo $X' \subsetneq X$ es satisfacible con respecto a $\Sigma_{1_E} \cup \Sigma_{1_{NC}} \subset \Sigma_1$ existe algún $\alpha \in X$ tal que $\alpha \notin \Sigma_1'$.

- **Éxito para Datos**: Prueba omitida debido a que es análoga a la prueba para **Éxito para Restricciones**.

- **Vacuidad para Restricciones**: Si no existe $X \subset \Sigma_{1_T}$ tal que X es satisfacible con respecto a $\Sigma_{1_{NC}} \cup \Sigma_{1_E} \subset \Sigma_1$ y $X \cup \alpha$ es insatisfacible con respecto a $\Sigma_{1_{NC}} \cup \Sigma_{1_E} \subset \Sigma_1$, entonces $\alpha \in \Sigma_1'$.

Sea α tal que no existe $X \subset \Sigma_{1_T}$ donde X es satisfacible con respecto a $\Sigma_{1_{NC}} \cup \Sigma_{1_E} \subset \Sigma_1$ y $X \cup \alpha$ es insatisfacible con respecto a $\Sigma_{1_{NC}} \cup \Sigma_{1_E} \subset \Sigma_1$. Entonces, por Definición 5.1 vale que $\alpha \notin X$ para todo $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$. De la Proposición 6.1 se sigue que si $\alpha \notin X$ para todo $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$, entonces $\alpha \notin Y$ para todo $Y \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$. Por lo tanto, $\alpha \notin \bigcup(\perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)})$. Por definición de ϕ tenemos entonces que $\alpha \notin \phi(KB_1)$, y por Definición 6.6 vale que $\alpha \in \Sigma_1'$.

- **Vacuidad para Datos**: Prueba omitida debido a que es análoga a la prueba para **Vacuidad para Restricciones**.

- **Mínima Pérdida de Información**: Si $KB' \subseteq KB_1$ es coherente y consistente, entonces se verifica que $\Upsilon(KB_1) \not\subset KB'$.

  Sea $KB' \subseteq KB_1$ tal que KB' es coherente y consistente, y sean $\mathcal{CF}_{\Sigma_1} = \Sigma_1 \setminus \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB)}$ y $\mathcal{CF}_{D_1} = D_1 \setminus \perp\!\!\!\perp\!\!\!\perp_{(D_1, KB)}$ los conjuntos de fórmulas que no están en ningún kernel en Σ_1 y D_1, respectivamente.

  Supongamos *por absurdo* que $\Upsilon(KB_1) \subset KB'$. Por definición de $\Psi_{\phi,\varphi}(KB_1)$ tenemos que $\phi(KB_1) \subseteq \bigcup(\perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB)})$, y que $\varphi(KB_1) \subseteq \bigcup(\perp\!\!\!\perp\!\!\!\perp_{(D_1, KB)})$. Entonces, $\mathcal{CF}_{\Sigma_1} \subseteq \Psi_{\phi,\varphi}(KB_1)$ y $\mathcal{CF}_{D_1} \subseteq \Psi_{\phi,\varphi}(KB_1)$. Por lo tanto, tenemos que $\mathcal{CF}_{\Sigma_1} \subset KB'$ y que $\mathcal{CF}_{D_1} \subset KB'$.

  Entonces, como $\Upsilon(KB_1) \subset KB'$ debe existir $\alpha \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB)} \cup \perp\!\!\!\perp\!\!\!\perp_{(D_1, KB)}$ tal que $\alpha \in KB'$ pero $\alpha \notin \Upsilon(KB_1)$, mientras que KB' es igualmente coherente y consistente. Esto es, hay una cluster de dependencias o un cluster de datos donde la elección en el cluster no es óptima, ya que se podría haber incluido a α en la consolidación. Para el resto de la prueba, por razones de simplicidad consideraremos el caso de que α es parte de un cluster de dependencias. Esto es hecho sin pérdida de generalidad, ya que la prueba para el caso donde α esta incluida en un cluster de datos es análoga a la que presentamos.

  Consideremos entonces $\alpha \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB)}$ tal que $\alpha \in KB'$. Por Corolario 6.2 tenemos que $\alpha \in X$ donde $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB)}$. Sea $T = (X \cap \phi(KB))$ la incisión realizada en el cluster, y sea $R = (X \cap \{KB \setminus KB'\})$ aquellas fórmulas eliminadas de X (*esto es,* la "incisión", aunque no haya sido necesariamente realizada a través de una función de incisión) en la obtención de KB'. Claramente, como KB' es coherente entonces para cada $Y \subseteq X$ donde $Y \in \perp\!\!\!\perp_{(\Sigma_1, KB)}$ vale que $R \cap Y \neq \emptyset$, ya que de lo contrario $Y \subset KB'$, lo que haría a KB' incoherente. Además, como $R \subseteq Y$

entonces $R \subseteq \perp\!\!\!\perp_{(\Sigma_1, KB)}$, y por lo tanto R satisface las dos primeras condiciones en la Definición 6.4.

A su vez, por Definición 6.4 tenemos que T es tal que no existe un conjunto de TGDs que satisfagan las dos primeras condiciones en la definición y a su vez R sea un subconjunto estricto del mismo, es decir que vale que $T \subset R$ (1).

Como $\alpha \notin \Upsilon(KB_1)$ y $\alpha \in X$ entonces $\alpha \in \phi(KB)$, y por lo tanto $\alpha \in T$. Sin embargo, sabemos que $\alpha \in X$ y $\alpha \in KB'$, y por lo tanto $\alpha \notin R$. Por lo tanto, tenemos que $T \not\subset R$ (2).

Por (1) y (2) tenemos que $T \subset R$ y que $T \not\subset R$, un absurdo que surge de nuestra suposición inicial de que $\Upsilon(KB_1) \subset KB'$, y vale que si $KB' \subseteq KB_1$ es coherente y consistente, entonces se verifica que $\Upsilon(KB_1) \not\subset KB'$.

$\Leftarrow$) Postulados a Construcción

Para la segunda parte de la prueba, considere un operador $\Psi_{\phi,\varphi}$ que satisface todos los postulados en el Teorema 6.1. Sea $\phi_{(\Psi)_\Sigma}$ un función definida en base a $\Psi_{\phi,\varphi}$ como sigue:

$$\phi_{(\Psi)_\Sigma}(KB_1) = \{x \,|\, x \in X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)} \textit{and}\, x \notin \{\Sigma_1 \cap \Psi_{\phi,\varphi}(KB_1)\}\}$$

Sea $KB_1^\star = (D_1, \Sigma_1 \setminus \phi_{(\Psi)_\Sigma}(KB_1))$ la ontología resultante de eliminar de KB_1 las TGDs seleccionadas por $\phi_{(\Psi)_\Sigma}$. Entonces, sea $\varphi_{(\Psi)_D}$ otra función definida en base a $\Upsilon_{\rho,\varrho}$ como sigue:

$$\varphi_{(\Psi)_D}(KB_1^\star) = \{x \,|\, x \in X \in \perp\!\!\!\perp_{(D_1, KB_1^\star)} \textit{and}\, x \notin \{D_1 \cap \Psi_{\phi,\varphi}(KB_1)\}\}$$

En base a $\varphi_{(\Psi)_D}$ y $\phi_{(\Psi)_\Sigma}$ definimos un nuevo operador como sigue:

$$\Psi'_{\phi,\varphi}(KB_1) = (D_1 \setminus \varphi_{(\Psi)_D}(KB_1^\star), \Sigma_1 \setminus \phi_{(\Psi)_\Sigma}(KB_1))$$

Tenemos que mostrar que $\Psi'_{\phi,\varphi}$ es un *operador de consolidación de ontologías Datalog$^\pm$ basado en Cluster Contraction*. Para hacer esto, primero mostramos que $\varphi_{(\Psi)_D}$ es una *función de incisión en clusters de datos* bien definida y que $\phi_{(\Psi)_\Sigma}$ es una *función de incisión en clusters de dependencias* bien definida. Esto es, dada $\phi_{(\Psi)_\Sigma}$ tenemos que probar que:

- $\phi_{(\Psi)_\Sigma}(KB_1) \subseteq \bigcup(\perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)})$.

  Esto es obtenido directamente de la definición de $\phi_{(\Psi)_\Sigma}$, ya que para todo $x \in \phi_{(\Psi)_\Sigma}(KB_1)$ se verifica que $x \in X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ debido a la primer condición de la definición.

- Si $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ e $Y \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ son tal que $Y \neq \emptyset$ e $Y \subseteq X$, entonces $(Y \cap \phi_{(\Psi)_\Sigma}(KB_1)) \neq \emptyset$.

  Supongamos *por absurdo* que existe algún $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ e $Y \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ tal que $Y \neq \emptyset$, $Y \subseteq X$ e $(Y \bigcap \phi_{(\Psi)_\Sigma}(KB_1)) = \emptyset$.

  Entonces, para todo $\alpha \in Y$ se verifica que $\alpha \notin \phi_{(\Psi)_\Sigma}(KB_1)$, *esto es,* $\alpha \notin Y \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ o $\alpha \in \{\Sigma_1 \cap \Psi_{\phi,\varphi}(KB_1)\}$. Por hipótesis tenemos que $\alpha \in Y \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$, y entonces $\alpha \in \{\Sigma_1 \cap \Psi_{\phi,\varphi}(KB_1)\}$, y por extensión $\alpha \in \Psi_{\phi,\varphi}(KB_1)$.

  Como $Y \subset \Sigma_1$, entonces $\alpha \in \Sigma_1$. Como $\alpha \in \Psi_{\phi,\varphi}(KB_1)$ y $\alpha \in \Sigma_1$, entonces por **Éxito para Restricciones** tenemos que existe algún $\beta \in Y$ tal que $\beta \notin \Psi_{\phi,\varphi}(KB_1)$. Entonces, $\beta \notin \{\Sigma_1 \cap \Psi_{\phi,\varphi}(KB_1)\}$. Por lo tanto, $\beta \in \phi_{(\Psi)_\Sigma}(KB_1)$; y $\beta \in (Y \bigcap \phi_{(\Psi)_\Sigma}(KB_1))$. Entonces, $(Y \bigcap \phi_{(\Psi)_\Sigma}(KB_1)) \neq \emptyset$.

  Por lo tanto tenemos que $(Y \bigcap \phi_{(\Psi)_\Sigma}(KB_1)) = \emptyset$ e $(Y \bigcap \phi_{(\Psi)_\Sigma}(KB_1)) \neq \emptyset$, un absurdo que surge de nuestra suposición inicial de que existe algún $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ e $Y \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ tal que $Y \neq \emptyset$, $Y \subseteq X$ e $(Y \bigcap \phi_{(\Psi)_\Sigma}(KB_1)) = \emptyset$, y se verifica que para todo $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ e $Y \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ tal que $Y \subseteq X$, si $Y \neq \emptyset$ entonces $(Y \bigcap \phi_{(\Psi)_\Sigma}(KB_1)) \neq \emptyset$.

- Para todo $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ se verifica que $T = (X \cap \phi_{(\Psi)_\Sigma}(KB_1))$ es tal que no existe $R \subset X$ donde R satisface las dos condiciones previas y $R \subsetneq T$.

  Para probar esto es suficiente con probar que, al ser los clusters disjuntos, la elección en cada cluster es optimal, ya que de lo contrario si existiera este cluster donde la función de incisión no elige optimalmente entonces no se satisfaciría **Mínima Pérdida de Información**. Supongamos *por absurdo* que existe $X \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ donde $T = (X \cap \phi_{(\Psi)_\Sigma}(KB_1))$ es tal que existe $R \subset X$ donde R satisface las dos condiciones previas y $R \subsetneq T$.

  Consideremos $KB' = (\Sigma', D')$ donde para todo $Y \in \perp\!\!\!\perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ tal que $Y \neq X$ se verifica que $T' = (Y \cap \phi_{(\Psi)_\Sigma}(KB_1))$ y $R' = (Y \cap \{KB \setminus KB'\})$ (aquellas fórmulas eliminadas de Y en la obtención de KB') son tales que $T' = R'$. Como $T' = R'$

entonces R' es tal que las condiciones en Definición6.4 son satisfechas. Además, sean $\mathcal{CF}_{\Sigma_1} = \Sigma_1 \setminus \perp\!\!\!\perp_{(\Sigma_1, KB)}$ y $\mathcal{CF}_{D_1} = D_1 \setminus \perp\!\!\!\perp_{(D_1, KB)}$ los conjuntos de fórmulas que no están en ningún kernel en Σ_1 y D_1, respectivamente; y sea KB' tal que $\mathcal{CF}_{\Sigma_1} \subset \Sigma'$ y $\mathcal{CF}_{D_1} \subset D'$.

El hecho de que todas las fórmulas que no se encuentran en conflictos pertenezcan a KB' y que KB' sea formada de manera tal que la elección en cada cluster distinto de X es igual tanto en KB' como en $\phi_{(\Psi)_\Sigma}(KB_1)$ hace que $KB' \setminus (X \cap \{KB \setminus KB'\}) = \Psi_{\phi,\varphi}(KB_1) \setminus (X \cap \phi_{(\Psi)_\Sigma}(KB_1))$. Esto es, si hay una diferencia entre KB' y $\Psi_{\phi,\varphi}(KB_1)$ la misma se debe a la elección de fórmulas eliminadas en el cluster X.

Finalmente, por suposición tenemos que existe $R \subset X$ donde R satisface las dos condiciones previas y $R \subsetneq T$. Sea KB' y $R \subsetneq T$ tal que $R = (X \cap \{KB \setminus KB'\})$ es el conjunto de fórmulas removidas de X en la construcción de KB'. Entonces, tenemos que KB' es coherente y consistente, ya que todo conflicto en los clusters en KB_1 fueron atendidos, ya sea removiendo R (para X) o los R' (para todo cluster distinto de X). Además, como tenemos que $KB' \setminus (X \cap \{KB \setminus KB'\}) = \Psi_{\phi,\varphi}(KB_1) \setminus (X \cap \phi_{(\Psi)_\Sigma}(KB_1))$ y que $R \subsetneq T$, entonces para $\Upsilon(KB_1) = KB_1 \setminus \phi_{(\Psi)_\Sigma}(KB_1)$ y $KB' = KB_1 \setminus \{ \bigcup_{Y \in \{\perp\!\!\!\perp_{(\Sigma_1, KB_1)} \setminus X\}} R' \cup R\}$ donde $(R' = Y \cap \{KB \setminus KB'\})$ y $R = (X \cap \{KB \setminus KB'\})$ vale que $\Upsilon(KB_1) \subset KB'$ (1). Esto es, si todas las fórmulas que no están en conflictos pertenecen tanto a $\Upsilon(KB_1)$ como a KB', en cada cluster distinto de X se remueven las mismas fórmulas, y el conjunto de fórmulas removidas de X para formar KB' son un subconjunto estricto de aquellas removidas por $\phi_{(\Psi)_\Sigma}(KB_1)$ para formar $\Upsilon(KB_1)$, entonces $\Upsilon(KB_1)$ es un subconjunto estricto de KB', *esto es,* eliminamos más fórmulas eliminando T que eliminando R.

Por otro lado, como KB' es coherente y consistente, entonces por **Mínima Pérdida de Información** tenemos que $\Upsilon(KB_1) \not\subset KB'$ (2). Por lo tanto, de (1) y (2) tenemos que $\Upsilon(KB_1) \subset KB'$ y $\Upsilon(KB_1) \not\subset KB'$, un absurdo que surge de nuestra suposición inicial de que existe $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ donde $T = (X \cap \phi_{(\Psi)_\Sigma}(KB_1))$ es tal que existe $R \subset X$ donde R satisface las dos condiciones previas y $R \subsetneq T$, y vale que para todo $X \in \perp\!\!\!\perp_{(\Sigma_1, KB_1)}$ se verifica que $T = (X \cap \phi_{(\Psi)_\Sigma}(KB_1))$ es tal que no existe $R \subset X$ donde R satisface las dos condiciones previas y $R \subsetneq T$.

Omitiremos la prueba de que $\varphi_{(\Psi)_D}$ es una *función de incisión en datos* usando **Éxito para Datos**, **Vacuidad para Datos** y **Mínima Pérdida de Información** ya que la

misma es análoga a la prueba de que $\phi_{(\Psi)_\Sigma}$ es una *función de incisión en dependencias* bien definida usando **Éxito para Restricciones**, **Vacuidad para Restricciones** y **Mínima Pérdida de Información**.

Para finalizar esta segunda parte de la prueba, ahora que hemos probado que $\varphi_{(\Psi)_D}$ y $\phi_{(\Psi)_\Sigma}$ son una *función de incisión en clusters de datos* y una *función de incisión en clusters de dependencias* respectivamente, tenemos que probar que $\Psi'_{\phi,\varphi}$ coincide con $\Psi_{\phi,\varphi}$. De **Inclusión** se sigue que $D'_1 \subseteq D_1$ y $\Sigma'_1 \subseteq \Sigma_1$ (1). Ademas, de nuestra definición de $\varphi_{(\Psi)_D}$ se sigue que $\varphi_{(\Psi)_D}(KB_1^\star) = D_1 \setminus D'_1$, y de nuestra definición de $\phi_{(\Psi)_\Sigma}$ se sigue que $\phi_{(\Psi)_\Sigma}(KB_1) = \Sigma_1 \setminus \Sigma'_1$ (2).

Entonces, por (1) y (2) tenemos que $D'_1 = D_1 \setminus \varphi_{(\Psi)_D}(KB_1^\star)$ y $\Sigma'_1 = \Sigma_1 \setminus \phi_{(\Psi)_\Sigma}(KB_1)$. Por lo tanto,

$$\Psi_{\phi,\varphi} = (D_1 \setminus \varphi_{(\Psi)_D}(KB_1^\star), \Sigma_1 \setminus \phi_{(\Psi)_\Sigma}(KB_1))$$

y entonces $\Psi'_{\phi,\varphi}$ coincide con $\Psi_{\phi,\varphi}$. ∎

Corolario 6.4 [página 137] *$\Psi_{\phi,\varphi}$ satisface **Coherencia** y **Consistencia**.*

Prueba Se sigue de la Proposición 5.1 que si un operador satisface **Inclusión** y **Éxito para Datos** entonces satisface **Consistencia**, y si satisface **Inclusión** y **Éxito para Restricciones** entonces satisface **Coherencia**. ∎

Bibliografía

[ABC99] Arenas, M., Bertossi, L. E., and Chomicki, J. Consistent query answers in inconsistent databases. In *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems (PODS '99)* (1999), pp. 68–79.

[ADB07] Arieli, O., Denecker, M., and Bruynooghe, M. Distance semantics for database repair. *Annals of Mathematics and Artificial Intelligence (AMAI) 50*, 3-4 (2007), 389–415.

[AFM06] Andritsos, P., Fuxman, A., and Miller, R. J. Clean answers over dirty databases: A probabilistic approach. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)* (2006), p. 30.

[AGM85] Alchourrón, C., Gärdenfors, P., and Makinson, D. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic 50*, 2 (1985), 510–530.

[AK05] Amgoud, L., and Kaci, S. An argumentation framework for merging conflicting knowledge bases: The prioritized case. In *Proceedings of 8th European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQUARU 2005)* (2005), pp. 527–538.

[AM81] Alchourrón, C., and Makinson, D. Hierarchies of Regulation and Their Logic. *New Studies in Deontic Logic* (1981), 125–148.

[AM85] Alchourrón, C., and Makinson, D. On the Logic of Theory Change: Safe Contraction. *Studia Logica 44* (1985), 405–422.

[Baa03] Baader, F. *The description logic handbook: theory, implementation, and applications*. Cambridge university press, 2003.

[BB97] Beneventano, D., and Bergamaschi, S. Incoherence and subsumption for recursive views and queries in object-oriented data models. *Data and Knowledge Engineering 21*, 3 (1997), 217–252.

[BCM+03] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., Eds. *The Description Logic Handbook: Theory, Implementation, and Applications* (2003), Cambridge University Press.

[BFFR05] Bohannon, P., Flaster, M., Fan, W., and Rastogi, R. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the 31st ACM SIGMOD international conference on Management of Data (ACM SIGMOD 2005)* (2005), pp. 143–154.

[BH08] Bernstein, P. A., and Haas, L. M. Information integration in the enterprise. *Communications of the ACM 51*, 9 (2008), 72–79.

[BHP09] Black, E., Hunter, A., and Pan, J. Z. An argument-based approach to using multiple ontologies. In *Proceedings of 3rd International Conference on Scalable Uncertainty Management (SUM 2009)* (2009), pp. 68–79.

[BKM91] Baral, C., Kraus, S., and Minker, J. Combining multiple knowledge bases. *IEEE Transactions on Knowledge and Data Engineering (TKDE) 3*, 2 (1991), 208–220.

[BLHL01] Berners-Lee, T., Hendler, J., and Lassila, O. The semantic web. *Scientific American 284(5):3443* (2001).

[BMVW10] Booth, R., Meyer, T. A., Varzinczak, I. J., and Wassermann, R. Horn belief change: A contraction core. In *Proceedings of 19th European Conference on Artificial Intelligence (ECAI 2010)* (2010), pp. 1065–1066.

[Bor95] Borgida, A. Description logics in data management. *IEEE Transactions on Knowledge and Data Engineering (TKDE) 7*, 5 (1995), 671–682.

[BQL07] Bell, D. A., Qi, G., and Liu, W. Approaches to inconsistency handling in description-logic based ontologies. In *On the Move to Meaningful Internet Systems (OTM Workshops (2))* (2007), pp. 1303–1311.

[BR13] Bienvenu, M., and Rosati, R. Tractable approximations of consistent query answering for robust ontology-based data access. In *Proceedings of 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)* (2013), pp. 775–781.

[Bra01] Bratko, I. *Prolog programming for artificial intelligence*. Pearson education, 2001.

[BS98] Besnard, P., and Schaub, T. Signed systems for paraconsistent reasoning. *Journal of Automated Reasoning 20*, 1 (1998), 191–213.

[CDL01] Calvanese, D., De Giacomo, G., and Lenzerini, M. A framework for ontology integration. In *Proceedings of 1st Semantic Web Working Symposium (SWWS'01)* (2001), I. F. Cruz, S. Decker, J. Euzenat, and D. L. McGuinness, Eds., pp. 303–316.

[CFS06] Cecchi, L., Fillottrani, P., and Simari, G. R. On the complexity of delp through game semantics. In *Proceedings of 8th International Workshop on Non-Monotonic Reasoning (NMR 2006)* (2006), J. Dix and A. Hunter, Eds., pp. 386–394.

[CGK08] Calì, A., Gottlob, G., and Kifer, M. Taming the infinite chase: Query answering under expressive relational constraints. In *Proceedings of 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)* (2008), pp. 70–80.

[CGK13] Calì, A., Gottlob, G., and Kifer, M. Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research (JAIR) 48* (2013), 115–174.

[CGL12] Calì, A., Gottlob, G., and Lukasiewicz, T. A general Datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web 14* (2012), 57–83.

[CGP12] Calì, A., Gottlob, G., and Pieris, A. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence 193* (2012), 87–128.

[CGZ09] Caroprese, L., Greco, S., and Zumpano, E. Active integrity constraints for database consistency maintenance. *IEEE Transactions on Knowledge and Data Engineering 21*, 7 (2009), 1042–1058.

[Cho98] Cholvy, L. Reasoning about merged information. In *Belief Change*, D. Dubois, H. Prade, D. M. Gabbay, and P. Smets, Eds., vol. 3 of *Handbook of Defeasible Reasoning and Uncertainty Management Systems*. Springer Netherlands, 1998, pp. 233–263.

[CKPR73] Colmeraner, A., Kanoui, H., Pasero, R., and Roussel, P. Un systeme de communication homme-machine en francais. Luminy.

[CMR12] Coronel, C., Morris, S., and Rob, P. *Database systems*. Independence, KY: Cengage, 2012.

[CT11] Caroprese, L., and Truszczynski, M. Active integrity constraints and revision programming. *Theory and Practice of Logic Programming 11*, 6 (2011), 905–952.

[Dal88] Dalal, M. Investigations into a theory of knowledge base revision. In *Proceedings of 7th National Conference on Artificial Intelligence (AAAI 1988)* (1988), pp. 475–479.

[Das92] Das, S. K. Deductive databases and logic programming.

[Dat12] Date, C. *Database Design and Relational Theory: Normal Forms and All That Jazz*. .o'Reilly Media, Inc.", 2012.

[dCBB95] da Costa, N. C. A., Béziau, J.-Y., and Bueno, O. A. S. Aspects of paraconsistent logic. *Logic Journal of the Interest Group in Pure and Applied Logic (IGPL) 3*, 4 (1995), 597–614.

[DDL06] Delgrande, J. P., Dubois, D., and Lang, J. Iterated revision as prioritized merging. In *Proceedings of 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)* (2006), pp. 210–220.

[Del11] Delgrande, J. P. Revising by an inconsistent set of formulas. In *Proceedings of 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)* (2011), pp. 833–838.

[DFDG+13] Deagustini, C. A. D., Fulladoza Dalibón, S. E., Gottifredi, S., Falappa, M. A., Chesñevar, C. I., and Simari, G. R. Relational databases as a massive information source for defeasible argumentation. *Knowledge-Based Systems 51* (2013), 91–109.

[DHI12] Doan, A., Halevy, A., and Ives, Z. *Principles of data integration*. Elsevier, 2012.

[DJ12] Delgrande, J. P., and Jin, Y. Parallel belief revision: Revising by sets of formulas. *Artificial Intelligence 176*, 1 (2012), 2223–2245.

[DMFS14a] Deagustini, C. A. D., Martinez, M. V., Falappa, M. A., and Simari, G. R. Inconsistency resolution and global conflicts. In *Proceedings of 21st European Conference on Artificial Intelligence (ECAI 2014)* (2014), pp. 991–992.

[DMFS14b] Deagustini, C. A. D., Martinez, M. V., Falappa, M. A., and Simari, G. R. Improving inconsistency resolution by considering global conflicts. In *Proceedngs of 8th International Conference on Scalable Uncertainty Management (SUM 2014)* (2014), pp. 120–133.

[DMFS15a] Deagustini, C. A. D., Martinez, M. V., Falappa, M. A., and Simari, G. R. On the influence of incoherence in inconsistency-tolerant semantics for datalog+/-. In *Proceedings of Ontologies and Logic Programming for Query Answering (ONTOLP 2015), Workshop of 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)* (2015).

[DMFS15b] Deagustini, C. A. D., Martinez, M. V., Falappa, M. A., and Simari, G. R. Datalog+/- ontology consolidation. *Journal of Artificial Intelligence Research (JAIR)* (2015). En revision.

[DSTW09] Delgrande, J. P., Schaub, T., Tompits, H., and Woltran, S. Merging logic programs under answer set semantics. In *Proceedings of 25th International Conference on Logic Programming (ICLP 2009)* (2009), pp. 160–174.

[DW09] Dunne, P., and Wooldridge, M. *Argumentation in Artificial Intelligence*. Springer, 2009, ch. Complexity of Abstract Argumentation, pp. 85–104.

[EE01] Enderton, H., and Enderton, H. B. *A mathematical introduction to logic*. Academic press, 2001.

[EKM08] Everaere, P., Konieczny, S., and Marquis, P. Conflict-based merging operators. In *Proceedings of 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)* (2008), pp. 348–357.

[Fal99] Falappa, M. A. *Teoría de Cambio de Creencias y sus Aplicaciones sobre Estados de Conocimiento*. PhD thesis, Universidad Nacional del Sur, 1999.

[Fen01] Fensel, D. Ontologies: Dynamic networks of formally represented meaning. *Vrije University: Amsterdam* (2001).

[FGKIS13] Falappa, M. A., García, A. J., Kern-Isberner, G., and Simari, G. R. Stratified belief bases revision with argumentative inference. *Journal of Philosophical Logic 42*, 1 (2013), 161–193.

[FH99] Friedman, N., and Halpern, J. Y. Belief revision: A critique. *Journal of Logic, Language and Information 8*, 4 (1999), 401–420.

[FHP+06] Flouris, G., Huang, Z., Pan, J. Z., Plexousakis, D., and Wache, H. Inconsistencies, negations and changes in ontologies. In *Proceedings of 21st National Conference on Artificial Intelligence (AAAI 2006)* (2006), pp. 1295–1300.

[FKIRS12] Falappa, M. A., Kern-Isberner, G., Reis, M., and Simari, G. R. Prioritized and non-prioritized multiple change on belief bases. *Journal of Philosophical Logic 41*, 1 (2012), 77–113.

[FKMP03] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. Data exchange: Semantics and query answering. In *Proceedings of 9th International Conference on Database Theory (ICDT 2003)* (2003), pp. 207–224.

[FKMP05] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. Data exchange: semantics and query answering. *Theoretical Computer Science 336*, 1 (2005), 89–124.

[Fuh91] Fuhrmann, A. Theory contraction through base contraction. *Journal of Philosophical Logic 20* (1991), 175–203.

[Fuh97] Fuhrmann, A. *An Essay on Contraction*. CSLI, 1997.

[Gär82] Gärdenfors, P. Rule for rational changes of belief. *Philosophical Essay Dediccated To Lennart Åqvist on his Fiftieth Birthday* (1982), 88–101.

[Gär88] Gärdenfors, P. *Knowledge in Flux: Modeling the dynamics of epistemic states*. MIT Press, 1988.

[Gär03] Gärdenfors, P. *Belief revision*, vol. 29. Cambridge University Press, 2003.

[GCS08] Gómez, S. A., Chesnevar, C. I., and Simari, G. R. An argumentative approach to reasoning with inconsistent ontologies. In *Proceedings of the KR Workshop on Knowledge Representation and Ontologies* (2008), pp. 11–20.

[GCS10] Gómez, S. A., Chesñevar, C. I., and Simari, G. R. Reasoning with inconsistent ontologies through argumentation. *Applied Artificial Intelligence 24*, 1&2 (2010), 102–148.

[Gel08] Gelfond, M. Answer sets. In *Handbook of Knowledge Representation*, F. van Harmelen, V. Lifschitz, and B. Porter, Eds., Foundations of Artificial Intelligence. Elsevier, 2008, ch. 7, pp. 285–316.

[GGS08] Gottifredi, S., García, A. J., and Simari, G. R. Defeasible knowledge and argumentative reasoning for 3APL agent programming. In *Proccedings of 12th International Workshop on Non-Monotonic Reasoning (NMR 2008)* (2008).

[GL90] Gelfond, M., and Lifschitz, V. Logic programs with classical negation, logic programming, 1990.

[GM88] Gärdenfors, P., and Makinson, D. Revisions of knowledge systems using epistemic entrenchment. In *Proceedings of the 2nd conference on Theoretical aspects of reasoning about knowledge (TARK 1988)* (1988), Morgan Kaufmann Publishers Inc., pp. 83–95.

[GM12] Greco, S., and Molinaro, C. Probabilistic query answering over inconsistent databases. *Annals of Mathematics and Artificial Intelligence (AMAI) 64*, 2-3 (2012), 185–207.

[GR92] Gärdenfors, P., and Rott, H. Belief revision. Tech. rep., Lund University Cognitive Studies, 1992.

[Gro88] Grove, A. Two modellings for theory change. *Journal of philosophical logic 17*, 2 (1988), 157–170.

[GS04] García, A. J., and Simari, G. R. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming 4*, 1-2 (2004), 95–138.

[Han89] Hansson, S. New operators for theory change. *Theoria 55*, 2 (1989), 114–132.

[Han91a] Hansson, S. O. *Belief Base Dynamics*. PhD thesis, Uppsala University, Department of Philosophy, Uppsala, Sweden, 1991.

[Han91b] Hansson, S. O. Belief contraction without recovery. *Studia Logica 50*, 2 (1991), 251–260.

[Han92] Hansson, S. O. A dyadic representation of belief. *Belief revision 29* (1992), 89–121.

[Han93] Hansson, S. O. Theory contraction and base contraction unified. *Journal of Symbolic Logic 58*, 2 (1993), 602–625.

[Han94] Hansson, S. O. Kernel contraction. *Journal of Symbolic Logic 59*, 3 (1994), 845–859.

[Han97a] *Theoria: Special Issue on Non-Prioritized Belief Revision*. Department of Philosophy, Uppsala University, 1997.

[Han97b] Hansson, S. O. Semi-revision (invited paper). *Journal of Applied Non-Classical Logics 7*, 2 (1997).

[Han99] Hansson, S. O. *A Textbook of Belief Dynamics: Theory Change and Database Updating*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[Han01] Hansson, S. O. *A Textbook of Belief Dynamics: Solutions to Exercises*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[Har75] Harper, W. Rational Belief Change, Popper Functions and Counterfactuals. *Synthese 30* (1975), 221–262.

[HFCF01] Hansson, S. O., Fermé, E. L., Cantwell, J., and Falappa, M. A. Credibility limited revision. *The Journal of Symbolic Logic 66*, 04 (2001), 1581–1596.

[HK06] Halaschek-Wiener, C., and Katz, Y. Belief base revision for expressive description logics. In *Proceedings of 2nd international workshop on "OWL: Experiences and Directions" (OWLED 2006)* (2006).

[HM10] Halpin, T., and Morgan, T. *Information modeling and relational databases*. Morgan Kaufmann, 2010.

[Hor05a] Horrocks, I. OWL: A description logic based ontology language. In *Proceedings of 11th International Conference on Principles and Practice of Constraint Programming (CP 2005)* (2005), P. van Beek, Ed., vol. 3709 of *Lecture Notes in Computer Science*, Springer, pp. 5–8.

[Hor05b] Horrocks, I. OWL: A description logic based ontology language. In *Proceedings of 21st International Conference on Logic Programming (ICLP 2005)* (2005), M. Gabbrielli and G. Gupta, Eds., vol. 3668 of *Lecture Notes in Computer Science*, Springer, pp. 1–4.

[HPW09] Hué, J., Papini, O., and Würbel, E. Merging belief bases represented by logic programs. In *Proccedings of 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2009)* (2009), pp. 371–382.

[HR95] Hansson, S. O., and Rott, H. How not to change the theory of theory change: A reply to Tennant. *The British Journal for the Philosophy of Science 46(3)* (1995), 361–380.

[HRWL84] Hayes-Roth, F., Waterman, D., and Lenat, D. *Building expert systems*. Addison-Wesley, Reading, MA, 1984.

[HS05] Haase, P., and Stojanovic, L. Consistent evolution of OWL ontologies. In *The Semantic Web: Research and Applications*. Springer, 2005, pp. 182–197.

[HvHH+05] Haase, P., van Harmelen, F., Huang, Z., Stuckenschmidt, H., and Sure, Y. A framework for handling inconsistency in changing ontologies. In *Proceedings of 4th International Semantic Web Conference (ISWC 2005)* (2005), pp. 353–367.

[HvHtT05] Huang, Z., van Harmelen, F., and ten Teije, A. Reasoning with inconsistent ontologies. In *Proceedings of 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)* (2005), pp. 454–459.

[Ino91] Inoue, K. Extended logic programming with default assumptions. In *Proceedings of 8th international conference on Logic Programming (ICLP 1991)* (1991).

[JK84] Johnson, D. S., and Klug, A. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences 28*, 1 (1984), 167–189.

[Kal06] Kalyanpur, A. *Debugging And Repair Of OWL Ontologies*. PhD thesis, 2006.

[KM91] Katsuno, H., and Mendelzon, A. O. On the difference between updating a knowledge base and revising it. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR 1991)* (1991), pp. 387–394.

[KM92] Katsuno, H., and Mendelzon, A. O. Propositional knowledge base revision and minimal change. *Artificial Intelligence 52*, 3 (1992), 263–294.

[Kon00] Konieczny, S. On the difference between merging knowledge bases and combining them. In *Proceedings of 7th International Conference on Principles of Knowledge Representation and Reasoning (KR2000)* (2000), pp. 135–144.

[Kow74] Kowalski, R. Predicate logic as programming language. In *IFIP congress* (1974), vol. 74, pp. 569–544.

[KP02] Konieczny, S., and Pérez, R. P. Merging information under constraints: A logical framework. *Journal of Logic and Computation 12*, 5 (2002), 773–808.

[KP11] Konieczny, S., and Pérez, R. P. Logic based merging. *Journal of Philosophical Logic 40*, 2 (2011), 239–270.

[KPSH05] Kalyanpur, A., Parsia, B., Sirin, E., and Hendler, J. A. Debugging unsatisfiable classes in owl ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web 3*, 4 (2005), 268–293.

[KS91] Kowalski, R. A., and Sadri, F. Logic programs with exceptions. *New Generation Computing 9*, 3-4 (1991), 387–400.

[Lev77] Levi, I. Subjunctives, Dispositions and Chances. *Synthese 34*, 4 (1977), 423–455.

[Lew73] Lewis, D. *Counterfactuals*. Harvard University Press, Cambridge, Massachusetts, 1973.

[Lif96] Lifschitz, V. Foundations of logic programs. In *Principles of Knowledge Representation*, G. Brewka, Ed. CSLI Pub., 1996, pp. 69–128.

[Llo87] Lloyd, J. W. *Foundations of Logic Programmming*. Springer-Verlag, 1987.

[LLR+10] Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., and Savo, D. F. Inconsistency-tolerant semantics for description logics. In *Proceedings of 4th International Conference on Web Reasoning and Rule Systems (RR 2010)* (2010), pp. 103–117.

[LM98] Lin, J., and Mendelzon, A. O. Merging databases under constraints. *International Journal of Cooperative Information Systems 7*, 1 (1998), 55–76.

[LM99] Lin, J., and Mendelzon, A. O. Knowledge base merging by majority. *Applied Logic Series 18* (1999), 195–218.

[LMS12] Lukasiewicz, T., Martinez, M. V., and Simari, G. I. Inconsistency handling in datalog+/- ontologies. In *Proceedings of 20th European Conference on Artificial Intelligence (ECAI 2012)* (2012), pp. 558–563.

[LS98] Liberatore, P., and Schaerf, M. Arbitration (or how to merge knowledge bases). *Knowledge and Data Engineering, IEEE Transactions 10*, 1 (1998), 76–90.

[MDFS14] Martinez, M. V., Deagustini, C. A. D., Falappa, M. A., and Simari, G. R. Inconsistency-tolerant reasoning in datalog± ontologies via an argumentative semantics. In *Proceedings of 14th Ibero-American Conference on AI, Advances in Artificial Intelligence (IBERAMIA 2014)* (2014), pp. 15–27.

[MLB05] Meyer, T., Lee, K., and Booth, R. Knowledge integration for description logics. In *Proceedings of 7th International Symposium on Logical Formalizations of Commonsense Reasoning (COMMONSENSE 2005)* (2005), AAAI Press, pp. 645–650.

[MLH00] Meyer, T. A., Labuschagne, W. A., and Heidema, J. Refined epistemic entrenchment. *Journal of Logic, Language and Information 9*, 2 (2000), 237–259.

[MMS79] Maier, D., Mendelzon, A. O., and Sagiv, Y. Testing implications of data dependencies. *ACM Transactions on Database Systems (TODS) 4*, 4 (1979), 455–469.

[MPP+14] Martinez, M. V., Parisi, F., Pugliese, A., Simari, G., and Subrahmanian, V. Policy-based inconsistency management in relational databases. *International Journal of Approximate Reasoning (IJAR) 55*, 2 (2014), 501 – 528.

[MPS+07] Martinez, M., Pugliese, A., Simari, G., Subrahmanian, V., and Prade, H. How dirty is your relational database? an axiomatic approach. In *Proceedings of 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2007)* (2007), pp. 103–114.

[MWF10] Moguillansky, M. O., Wassermann, R., and Falappa, M. A. An argumentation machinery to reason over inconsistent ontologies. In *Proceedings of 12th Ibero-American Conference on AI, Advances in Artificial Intelligence (IBERAMIA 2010)* (2010), pp. 100–109.

[NM95] Nilsson, U., and Maluszynski, J. *Logic, Programming and Prolog (2ed)*. John Wiley & Sons Ltd., 1995.

[NM+01] Noy, N. F., McGuinness, D. L., et al. Ontology development 101: A guide to creating your first ontology, 2001.

[Nor] https://northwinddatabase.codeplex.com/.

[Nut88] Nute, D. Defeasible reasoning: a philosophical analysis in prolog. In *Aspects of Artificial Intelligence* (1988), pp. 251–288.

[Pat14] Patel-Schneider, P. F. Using description logics for RDF constraint checking and closed-world recognition. *CoRR abs/1411.4156* (2014).

[Pat15] Patel-Schneider, P. F. Using description logics for RDF constraint checking and closed-world recognition. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.* (2015), B. Bonet and S. Koenig, Eds., AAAI Press, pp. 247–253.

[Pol87] Pollock, J. L. Defeasible reasoning. *Cognitive Science 11*, 4 (1987), 481–518.

[Pra10] Prakken, H. An abstract framework for argumentation with structured arguments. *Argument and Computation 1* (2010), 93–124.

[PUSC12] Pérez-Urbina, H., Sirin, E., and Clark, K. Validating rdf with owl integrity constraints, 2012. `http://docs.stardog.com/icv/icv-specification.html`.

[PWA03] Parsons, S., Wooldridge, M., and Amgoud, L. Properties and complexity of some formal inter-agent dialogues. *Journal of Logic and Computation 13*, 3 (2003), 347–376.

[QH07] Qi, G., and Hunter, A. Measuring incoherence in description logic-based ontologies. In *Proceedings of 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC 2007)* (2007), pp. 381–394.

[QLB06] Qi, G., Liu, W., and Bell, D. A. Knowledge base revision in description logics. In *Proceedings of 10th European Conference on Logics in Artificial Intelligence (JELIA 2006)* (2006), pp. 386–398.

[Rei87] Reiter, R. A theory of diagnosis from first principles. *Artificial Intelligence 32(1)* (1987), 57–95.

[RGS07] Rotstein, N. D., García, A. J., and Simari, G. R. Reasoning from desires to intentions: A dialectical framework. In *Proceedings of 22nd National Conference on Artificial Intelligence (AAAI 2007)* (2007), pp. 136–141.

[Rib13] Ribeiro, M. M. *Belief Revision in Non-Classical Logics*. Springer Briefs in Computer Science. Springer, 2013.

[Ros11] Rosati, R. On the complexity of dealing with inconsistency in description logic ontologies. In *Proceedings of 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)* (2011), pp. 1057–1062.

[Rot92] Rott, H. Preferential belief change using generalized epistemic entrenchment. *Journal of Logic, Language and Information 1*, 1 (1992), 45–78.

[Rou75] Roussel, P. *PROLOG: Manuel de Reference et d'Utilisation*. Université d'Aix-Marseille II, 1975.

[SCM12] Staworko, S., Chomicki, J., and Marcinkowski, J. Prioritized repairing and consistent query answering in relational databases. *Annals of Mathematics and Artificial Intelligence (AMAI) 64*, 2-3 (2012), 209–246.

[SHCvH07] Schlobach, S., Huang, Z., Cornet, R., and van Harmelen, F. Debugging incoherent terminologies. *Journal of Automated Reasoning 39*, 3 (2007), 317–349.

[Sim89] Simari, G. R. *A Mathematical Treatment of Defeasible Reasoning and its Implementation*. PhD thesis, Washington University, Dep. of Comp. Science, December 1989.

[SL92] Simari, G. R., and Loui, R. P. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence 53*, 2-3 (1992), 125–157.

[SMCS08] Sagui, F. M., Maguitman, A. G., Chesñevar, C. I., and Simari, G. R. Modeling news trust: A defeasible logic programming approach. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial 12*, 40 (2008), 63–72.

[SMMS02] Stojanovic, L., Maedche, A., Motik, B., and Stojanovic, N. User-driven ontology evolution management. In *Knowledge engineering and knowledge management: ontologies and the semantic web*. Springer, 2002, pp. 285–300.

[SS94] Sterling, L., and Shapiro, E. Y. *The art of Prolog: advanced programming techniques*. MIT press, 1994.

[Sto04] Stojanovic, L. *Methods and tools for ontology evolution.* PhD thesis, Karlsruhe Institute of Technology, 2004.

[Tam10] Tamargo, L. H. *Dinámica de Conocimiento en Sistemas Multi-Agentes: Plausibilidad, Revisión de Creencias y Retransmisión de Información*. PhD thesis, Universidad Nacional del Sur, 2010.

[TGGS14] Teze, J. C., Gottifredi, S., García, A. J., and Simari, G. R. An approach to argumentative reasoning servers with multiple preference criteria. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial 17*, 53 (2014), 68–78.

[Tur03] Turner, H. Strong equivalence made easy: nested expressions and weight constraints. *Theory and Practice of Logic Programming 3*, 4-5 (2003), 609–622.

[VEK76] Van Emden, M. H., and Kowalski, R. A. The semantics of predicate logic as a programming language. *Journal of the ACM (JACM) 23*, 4 (1976), 733–742.

[vL76] von Leibniz, G. W. F. *Philosophical Papers and Letters: a selection*, vol. 1. Springer, 1976.

[Was99] Wassermann, R. Resource bounded belief revision. *Erkenntnis 50*, 2-3 (1999), 429–446.

[Was00] Wassermann, R. An algorithm for belief revision. In *Procedings of 7th International Conference on Principles of Knowledge Representation and Reasoning (KR2000)* (2000), pp. 345–352.

[Wij05] Wijsen, J. Database repairing using updates. *ACM Transactions on Database Systems (TODS) 30*, 3 (September 2005), 722–768.