Universidad Nacional del Sur

Tesis de Doctor en Ciencias de la Computación

Integración de Argumentación Rebatible y Ontologías en el Contexto de la Web Semántica: Formalización y Aplicaciones

Sergio Alejandro Gómez

Bahía Blanca — Argentina

2009

Universidad Nacional del Sur

Tesis de Doctor en Ciencias de la Computación

Integración de Argumentación Rebatible y Ontologías en el Contexto de la Web Semántica: Formalización y Aplicaciones

Sergio Alejandro Gómez

Bahía Blanca — Argentina

2009

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 7 de septiembre de 2004 y el 15 de marzo de 2009, bajo la dirección de los Dres. Carlos Iván Chesñevar y Guillermo Ricardo Simari, Profesores del Departamento de Ciencias e Ingeniería de la Computación.

Mg. Sergio Alejandro Gómez
sag@cs.uns.edu.ar
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca, 15 de marzo de 2009.

UNIVERSIDAD NACIONAL DEL SUR
Secretaría General de Posgrado y Educación Continua

La presente tesis ha sido aprobada el .../.../..., mereciendo

la calificación de ..... (.....)

Agradecimientos

En primer lugar, deseo agradecer a mis directores Dres. Carlos Iván Chesñevar y Guillermo Ricardo Simari por el apoyo y la guía brindada a lo largo del desarrollo de la Tesis.

Quisiera además expresar mi agradecimiento al Departamento de Ciencias e Ingeniería de la Computación de la Universidad Nacional del Sur y especialmente a mis compañeros del Laboratorio de Investigación y Desarrollo en Inteligencia Artificial (LIDIA). Quisiera agradecer al equipo de programación de *DeLP Client*, en el cual fueron probados los ejemplos presentados en esta Tesis, y en particular a Nicolás Rotstein, quien me orientó en el uso del mismo. También quiero agradecer a Pablo Fillottrani por estar siempre dispuesto a discutir un problema referido a la Web Semántica y a brindar un consejo.

Agradezco a mi familia, en particular a mis padres, por el afecto, el hogar y la educación que me brindaron. Agradezco también a mi familia política por el apoyo que me dieron. Finalmente, agradezco a mi esposa, María Florencia Cittadini, por todo el amor, la paciencia, el apoyo en los momentos de incertidumbre y el tiempo brindado para que pudiera completar esta Tesis. Quiero agradecer también a mi hijo Tomás cuya inminente llegada al mundo sirvió para darme alegría y la motivación final para terminar la escritura de esta Tesis.

Gracias a la Agencia Nacional de Promoción Científica y Tecnológica por el apoyo económico que me brindó a través de una Beca de Doctorado en el período abril de 2004 a abril de 2007. También quiero agradecer sus orientaciones a los investigadores de la Universidad de Clausthal, Jürgen Dix y Wojtek Jamroga, con quienes pude interactuar a través de un proyecto bilateral SeCyT-DAAD. Finalmente quiero agradecer a los Jurados de esta Tesis por sus comentarios para mejorar la redacción de la misma.

Resumen

La World Wide Web actual está compuesta principalmente por documentos escritos para su presentación visual para usuarios humanos. Sin embargo, para obtener todo el potencial de la web es necesario que los programas de computadoras o agentes sean capaces de comprender la información presente en la web. En este sentido, la Web Semántica es una visión futura de la web donde la información tiene significado exacto, permitiendo así que las computadoras entiendan y razonen en base a la información hallada en la web. La Web Semántica propone resolver el problema de la asignación de semántica a los recursos web por medio de metadatos cuyo significado es dado a través de definiciones de ontologías, que son formalizaciones del conocimiento de un dominio de aplicación. El estándar del World Wide Web Consortium propone que las ontologías sean definidas en el lenguaje OWL, el cual se halla basado en las Lógicas para la Descripción. A pesar de que las definiciones de ontologías expresadas en Lógicas para la Descripción pueden ser procesadas por razonadores estándar, tales razonadores son incapaces de lidiar con ontologías inconsistentes.

Los sistemas argumentativos constituyen una formalización del razonamiento rebatible donde se pone especial énfasis en la noción de argumento. Así, la construcción de argumentos permite que un agente obtenga conclusiones en presencia de información incompleta y potencialmente contradictoria. En particular, la Programación en Lógica Rebatible es un formalismo basado en la argumentación rebatible y la Programación en Lógica.

En esta Disertación, la importancia de la definición de ontologías para poder llevar a cabo la realización de la iniciativa de la Web Semántica junto con la presencia de ontologías incompletas y potencialmente contradictorias motivó el desarrollo de un marco de razonamiento con las llamadas δ-ontologías. Investigaciones previas de otros autores, determinaron que un subconjunto de las Lógicas para la Descripción pueden ser traducidas efectivamente a un conjunto de la Programación en Lógica. Nuestra propuesta involucra asignar semántica a ontologías expresadas en Lógicas para la Descripción por medio de Programas Lógicos Rebatibles para lidiar con definiciones de ontologías inconsistentes en la Web Semántica. Esto es, dada una ontología Σ_{OWL} expresada en el lenguaje OWL-DL, es posible construir una ontología Σ_{DL} equivalente expresada en las Lógicas para la Descripción. En el caso en que Σ_{DL} satisfaga ciertas restricciones, ésta puede ser expresada como un programa DeLP $\mathcal{P}_\Sigma$. Por lo tanto, dada una consulta acerca de la pertenencia de una instancia a a un cierto concepto C expresada con respecto a Σ_{OWL}, se realiza un análisis dialéctico con respecto a $\mathcal{P}_\Sigma$ para determinar todas las razones a favor y en contra de la plausibilidad de la afirmación $C(a)$.

Por otro lado, la integración de datos es el problema de combinar datos residiendo en diferentes fuentes y el de proveer al usuario con una vista unificada de dichos datos. El problema de diseñar sistemas de integración de datos es particularmente importante en el contexto de aplicaciones en la Web Semántica donde las ontologías son desarrolladas independientemente unas de otras, y por esta razón pueden ser mutuamente inconsistentes. Dada una ontología, nos interesa conocer en qué condiciones un individuo es una instancia de un cierto concepto. Como cuando se tienen varias ontologías, los mismos conceptos pueden tener nombres distintos para un mismo significado o aún nombres iguales para significados diferentes, para relacionar los conceptos entre dos ontologías diferentes se utilizaron reglas puente o de articulación. De esta manera, un concepto se corresponde a una vista sobre otros conceptos de otra ontología. Mostramos también bajo qué condiciones la propuesta del razonamiento con δ-ontologías puede ser adaptada a los dos tipos de integración de ontologías *global-as-view* y *local-as-view* considerados en la literatura especializada.

Además, analizamos las propiedades formales que se desprenden de este acercamiento novedoso al tratamiento de ontologías inconsistentes en la Web Semántica. Los principales resultados obtenidos son que, como la interpretación de δ-ontologías como Programas Lógicos Rebatibles es realizada a través de una función de transformación que preserva la semántica de las ontologías involucradas, los resultados obtenidos al realizar consultas son sensatos. También, mostramos que el operador presentado es además consistente y significativo.

El acercamiento al razonamiento en presencia de ontologías inconsistentes brinda la posibilidad de abordar de una manera eficaz ciertos problemas de aplicación del ámbito del comercio electrónico, donde el modelo de reglas de negocio puede ser especificado en términos de ontologías. Entonces, la capacidad de razonar frente a ontologías inconsistentes permite abordajes alternativos conceptualmente más claros, ya que es posible automatizar ciertas decisiones de negocios tomadas a la luz de un conjunto de reglas de negocio posiblemente inconsistentes expresadas como una o varias ontologías y tener un sistema capaz de brindar una explicación del porqué se arribó a una conclusión determinada. En consecuencia, presentamos entonces una aplicación del razonamiento sobre ontologías inconsistentes por medio de la argumentación rebatible al modelado de formularios en la World Wide Web. La noción de los formularios como una manera de organizar y presentar datos ha sido utilizada desde el comienzo de la World Wide Web. Los formularios Web han evolucionado junto con el desarrollo de nuevos lenguajes de marcado, en los cuales es posible proveer guiones de validación como parte del código del formulario para verificar que el significado pretendido del formulario es correcto. Sin embargo, para el diseñador del formulario, parte de este significado pretendido frecuentemente involucra otras características que no son restricciones por sí mismas, sino más bien *atributos* emergentes del formulario, los cuales brindan conclusiones plausibles en el contexto de información incompleta y potencialmente contradictoria. Como el valor de tales atributos puede cambiar en presencia de nuevo conocimiento, los llamamos *atributos rebatibles*. Propusimos entonces extender los formularios web para incorporar atributos rebatibles como parte del conocimiento que puede ser codificado por el diseñador del formulario, por

medio de los llamados δ-formularios; dicho conocimiento puede ser especificado mediante un programa DeLP, y posteriormente, como una ontología expresada en Lógicas para la Descripción.

Palabras claves: Web, Web Semántica, Ontologías, Lógicas para la Descripción, Programación en Lógica Rebatible, argumentación rebatible, representación de conocimiento, Inteligencia Artificial.

Abstract

The current World Wide Web is based primarily on documents written with an emphasis on visual presentation. However, obtaining all the Web's potential requires enabling computer programs or agents to understand and reason on the basis of such information. In this sense, the Semantic Web is a future vision of the web where stored information has exact meaning, thus enabling computers to understand and reason on the basis of such information. Assigning semantics to web resources is addressed by means of metadata whose meaning is given by means of ontology definitions, *i.e.* formalizations of the knowledge in an application domain. The World Wide Web Consortium standard proposes that ontologies have to be defined in the OWL language, whose semantics are given in terms of Description Logics. Although ontology definitions expressed in Description Logics can be processed with existing reasoners, such reasoners are incapable of dealing with inconsistent ontology definitions.

Argumentation systems constitute a formalization of defeasible reasoning with special emphasis on the notion of argument. In this regard, the construction of arguments allows an agent to obtain conclusions in the presence of incomplete and potentially inconsistent information. In particular, Defeasible Logic Programming is a formalism based in defeasible argumentation and Logic Programming.

In this Dissertation, the importance of ontology definitions to carry out the realization of the Semantic Web initiative along with the presence of incomplete and potentially inconsistent ontologies motivated the development of a reasoning framework with the so-called δ-ontologies. Previous research have determined that a subset of Description Logics can be effectively translated into Logic Programming. Our proposal involves assigning semantics to Description Logics ontologies by means of Defeasible Logic Programming programs in order to deal with inconsistent ontology definitions in the Semantic Web. That is, given an ontology Σ_{OWL} expressed in the OWL-DL language, it is possible to build an equivalent Σ_{DL} ontology expressed in Description Logics. In the case that Σ_{DL} satisfies certain constraints, it can be expressed as a DeLP program $\mathcal{P}_{\Sigma}$. Therefore, given a query respect to the membership of an individual a to a certain concept C, a dialectical analysis with respect to $\mathcal{P}_{\Sigma}$ is carried out to determine all the reasons in favor and against the plausibility of $C(a)$.

On the other hand, data integration is referred to as the problem of combining data residing in different sources and providing the user with a unified view of such data. The problem of designing data integration systems is of particular importance in the context of Semantic Web applications where ontologies are developed independently from each

other, and for this reason they could be mutually inconsistent. Given an ontology, we are interested in determining under what conditions a certain individual is an instance of a certain concept. When we have several ontologies, same concepts can be named differently for a same meaning or even can have the same names for different meanings, therefore bridge or articulation rules are needed. Thus, a concept is regarded as a view over other concepts in some other ontology. We show under which conditions the δ-ontologies proposal can be adapted to the *global-as-view* and *local-as-view* ontology integration models present in the literature.

Besides, we analyze the formal properties arising from this novel approach to inconsistent ontologies in the Semantic Web. The main results obtained are that the interpretation of δ-ontologies as defeasible logic programs preserve the semantics of the involved ontologies. We also show that the entailment operator presented is sound, consistent and meaningful.

The approach to the reasoning in presence of inconsistent ontologies provides the possibility of tackling certain application problems in the electronic commerce domain, where the business rule model can be specified in terms of ontologies. Therefore, the ability of reasoning in the presence of inconsistent ontologies allows conceptually clearer alternative solutions, as it is possible to automatize certain business decisions made with respect to a possibly inconsistent set of business rules expressed as one or several ontologies and having a system capable of providing explanations of its conclusions. As a consequence, we present an application to reasoning with inconsistent ontologies by means of defeasible argumentation in the context of World Wide Web forms. The notion of forms as a way of organizing and presenting data has been used since the beginning of the World Wide Web. Web-based forms have evolved together with the development of new markup languages, in which it is possible to provide validation scripts as part of the form code to test whether the intended meaning of the form is correct. However, for the form designer, part of this intended meaning frequently involves other features which are not constraints by themselves, but rather *attributes* emerging from the form, which provide plausible conclusions in the context of incomplete and potentially inconsistent information. As the value of such attributes may change in presence of new knowledge, we call them *defeasible attributes*. In this Thesis, we propose extending traditional web-based forms to incorporate defeasible attributes as part of the knowledge that can be encoded by the form designer. The proposed extension allows the specification of scripts for reasoning about form fields using a defeasible knowledge base, expressed in terms of a Defeasible Logic Program, and a Description Logics ontology.

KEYWORDS: Web, Semantic Web, Ontologies, Description Logics, Defeasible Logic Programming, defeasible argumentation, knowledge representation, Artificial Intelligence

Publicaciones surgidas de esta tesis

En revistas internacionales:

- Sergio Alejandro Gómez, Carlos Iván Chesñevar and Guillermo Ricardo Simari. Reasoning with Inconsistent Ontologies Through Argumentation. Submitted to the *Journal of Applied Artificial Intelligence*. 2009.

- Sergio Alejandro Gómez, Carlos Iván Chesñevar and Guillermo Ricardo Simari. Defeasible Reasoning in Web-based Forms Through Argumentation. *International Journal of Information Technology & Decision Making (IJITDM)*. Volume 7, Issue 1 (March 2008), Pages 71–101, 2008.

- Sergio Alejandro Gómez, Carlos Iván Chesñevar and Guillermo Ricardo Simari. Inconsistent Ontology Handling by Translating Description Logics into Defeasible Logic Programming. *Revista Iberoamericana de Inteligencia Artificial. Editada por Asociación Española para la Inteligencia Artificial (AEPIA)*, Volumen 11, Número 35, Año 2007, Páginas 11-22. ISSN: 1137-3601. 2007.

En congresos internacionales:

- Sergio Alejandro Gómez, Carlos Iván Chesñevar and Guillermo Ricardo Simari. An Argumentative Approach to Reasoning with Inconsistent Ontologies. En *Proc. of the Knowledge Representation in Ontologies Workshop (KROW 2008)*. CPRIT 90. Pág. 11–20. Sydney, Australia, 2008. Thomas Meyer and Mehmet A. Orgun, eds.

- Sergio Alejandro Gómez, Carlos Iván Chesñevar and Guillermo Ricardo Simari. Incorporating Defeasible Knowledge and Argumentative Reasoning in Web-based Forms. Proc. of the *3rd Intl. Workshop on Intelligent Techniques for Web Personalization (ITWP 2005)*. In 19th Intl. Joint Conf. in Artificial Intelligence (IJCAI 2005). Pág. 9-16. Edimburgh, UK, July 2005.

En congresos nacionales:

- Sergio Alejandro Gómez, Carlos Iván Chesñevar and Guillermo Ricardo Simari. Integration of Web-based Forms with Ontologies in the Semantic Web. *Actas del XIV*

Congreso Argentino de Ciencias de la Computación (CACIC 2008). Chilecito, Argentina, 6-10 Octubre 2008 Compiladores: Norberto Caminoa, Fernanda Carmona, Antonio Castro Lechtaler 550 pp. ISBN 978-987-24611-0-2

- Sergio Alejandro Gómez, Carlos Iván Chesñevar and Guillermo Ricardo Simari. Hacia la Solución del Problema de Chequeo de Instancia en Ontologías Inconsistentes Usando Argumentación Rebatible. *Anales del Décimo Workshop de Investigadores en Ciencias de la Computación (WICC 2008)*. Páginas 66–70. General Pico, La Pampa, Argentina. 5 y 6 de mayo de 2008. ISBN 978-950-863-101-5.

- Sergio Alejandro Gómez, Carlos Iván Chesñevar and Guillermo Ricardo Simari. Hacia una Integración de Argumentación Rebatible y Ontologías en la Web Semántica. *Anales del Noveno Workshop de Investigadores en Ciencias de la Computación (WICC 2007)*. pág. 91–95, 2007.

- Sergio Alejandro Gómez, Carlos Iván Chesñevar and Guillermo Ricardo Simari. An Approach to Handling Inconsistent Ontology Definitions based on the Translation of Description Logics into Defeasible Logic Programming. *Proc. of the XII Congreso Argentino de Ciencias de la Computación (CACIC 2006)*. Potrero de Los Funes, San Luis, Argentina, 2006.

- Sergio Alejandro Gómez, Carlos Iván Chesñevar and Guillermo Ricardo Simari. Problems and Challenges for Ontology Integration in the Semantic Web. *Anales del Octavo Workshop de Investigadores en Ciencias de la Computación (WICC 2006)*. Pág. 69–73. Compilado por Jorge Salvador Ierache - 1a. ed. - Morón: Universidad de Morón, 2006. 751 p. ISBN 950-9474-34-7.

- Sergio Alejandro Gómez, Carlos Iván Chesñevar and Guillermo Ricardo Simari. A First Approach to Combining Ontologies and Defeasible Argumentation for the Semantic Web. Proc. of the *XI Congreso Argentino en Ciencias de la Computación (CACIC 2005)*. Concordia, Argentina. 2005.

- Sergio Alejandro Gómez, Carlos Iván Chesñevar and Guillermo Ricardo Simari. Embedding Defeasible Argumentation in the Semantic Web: an ontology-based approach. Proc. of the *VII Workshop de Investigadores en Ciencias de la Computación (WICC 2005)*, Río Cuarto, Argentina, Pp. 153-157, 2005.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción y motivaciones

La *World Wide Web* (WWW) actual está basada principalmente en documentos escritos para su presentación visual para usuarios humanos. Sin embargo, para obtener todo el potencial de la web es necesario que los programas de computadoras o *agentes* sean capaces de comprender la información en la web.

La *Web Semántica* (Berners-Lee et al., 2001) es una visión futura de la web donde la información tiene significado exacto, permitiendo así que las computadoras entiendan y razonen en base a la información hallada allí. Así, la Web Semántica propone resolver el problema de la asignación de semántica a los recursos web por medio de *metadatos* cuyo significado es dado a través de *definiciones de ontologías*.

En el contexto del área de compartir conocimiento, el término *ontología* se refiere a la especificación de una conceptualización; esto es, una ontología es una descripción de los conceptos y relaciones que pueden existir para un agente o una comunidad de agentes (Gruber, 1993). En particular, se tiene interés en que un agente inteligente, quizá inmerso en un *sistema multiagente*, pueda razonar a partir de esas ontologías y los datos que son descriptos por ellas, obteniendo así conclusiones *racionalmente justificadas* a partir de dichos datos. Se cree que el desarrollo de las tecnologías asociadas al programa de la Web Semántica tendrá un fuerte impacto tecnológico con beneficios como (Berners-Lee, 2003):

- *Búsqueda de documentos más eficiente*: Los buscadores de la web no tendrán que limitar sus enfoques a la aparición de un conjunto de claves de búsqueda en los documentos a revisar.
- *Validación en formularios web*: Muchas aplicaciones de la web utilizan interfaces de usuario basadas en formularios y los datos cargados por el usuario requieren validaciones complejas. Actualmente, éstas deben ser realizadas por métodos poco declarativos. Así, la definición de metadatos mejorará ostensiblemente la situación al permitir asociar significado a los valores de los campos de los formularios en forma no procedimental.

- *Distribución más eficaz de los datos*: Las caches de los proxies podrán determinar que los datos se distribuyen de acuerdo a los deseos de sus autores.

Por lo tanto, el concepto más importante de la Web Semántica es que los documentos son entendibles por las máquinas y no solamente por los usuarios. En consecuencia, agentes inteligentes con poder delegado de los usuarios pueden acceder a los datos y razonar acerca de su contenido para lograr los objetivos propuestos. De esta manera, la Web Semántica puede ser vista como la unión de dos grandes áreas de conocimiento: la Web, por un lado, y, por otro, el área de la representación de conocimiento (Davis et al., 1993).

Además, diversos autores (*e.g.*, véase la *Falacia 2* en (van Harmelen, 2006)) coinciden en el hecho de que el uso de una única ontología (por ejemplo, la ontología multi-contextual planteada por el proyecto *CyC* (Lenat, 1995)) no es una solución viable para la Web Semántica o, por ejemplo, para resolver el problema de la comunicación entre agentes en un sistema multiagente. Esta situación es debida a que cada agente tiene una visión diferente del mundo que lo rodea y dicha visión viene dada por los objetivos particulares de tal agente. Como dicha visión del mundo es descripta por una determinada ontología, en consecuencia cuando dos o más agentes interactúan en el contexto de un sistema multiagente, se plantea la necesidad de establecer una ontología común mediante las cuales los mismos se puedan comunicar y poner de acuerdo. Dicha actividad es conocida como *integración de ontologías*.

La comunidad científica ya ha encarado con un cierto éxito la solución de los objetivos del programa de la Web Semántica. Sin embargo, pensamos que todavía hay problemas sin resolver. En particular, en los enfoques actuales, las ontologías son descriptas en un *lenguaje de representación de ontologías*; dando más detalles, el *Lenguaje de Ontologías Web* (OWL) (McGuiness and van Harmelen, 2004) es el lenguaje propuesto por el estándar del Consorcio de la W3C para tal fin. Aunque dicho lenguaje provee un número expresivo de constructores, el mismo adolece del problema de no permitir razonar en presencia de información *incompleta* y potencialmente *contradictoria*.

En otro orden, la *Inteligencia Artificial* ha desarrollado mecanismos para abordar este último problema a través de distintas formalizaciones con el objetivo de encontrar un modelo adecuado para el razonamiento humano (también llamado *razonamiento del sentido común*). Uno de los enfoques más utilizados en tal sentido es el *razonamiento rebatible*, en el cual las conclusiones obtenidas por un agente inteligente pueden ser rechazadas ante la aparición de nueva evidencia.

Los *sistemas argumentativos* constituyen una formalización del razonamiento rebatible donde se pone especial énfasis en la noción de *argumento*. En este contexto, la construcción de argumentos permite que un agente obtenga conclusiones. Un *argumento* para una conclusión H constituye una pieza de razonamiento tentativa que un agente inteligente está dispuesto a aceptar para explicar H. Si el agente adquiriese luego nueva información, la conclusión H junto con el razonamiento que la produjo podrían quedar invalidados.

Dando más detalles, los sistemas argumentativos permiten formalizar adecuadamente el razonamiento de sentido común, donde las creencias y reglas de un agente pueden representarse a través de una teoría T expresada en un lenguaje lógico, que usualmente extiende la lógica de primer orden, y posibilita trabajar con reglas *rebatibles*. Un argumento $\mathcal{A}$ para una creencia C es un conjunto de reglas rebatibles (instanciadas) tales que $T \cup \mathcal{A}$ permite derivar a C como conclusión. Una nueva creencia C estará *garantizada* para nuestro agente inteligente sólo si existe un argumento $\mathcal{A}$ que sustente a C, y dicho argumento prevalezca por sobre otros *contraargumentos* que pudieran derrotarlo.

El objetivo perseguido en esta Tesis de Doctorado es el estudio de la integración de las distintas técnicas de la representación de conocimiento en la Web y la Web Semántica con los sistemas argumentativos. De esta manera, los resultados obtenidos son transferibles al problema del modelado del comportamiento de un agente racional que represente su conocimiento en base a ontologías y que utilice argumentación para determinar qué parte de su conocimiento está racionalmente 'garantizado' (en inglés *warranted*) (Cheong, 1995; Hunhs and Singh, 1998; Woolridge, 2002).

Se buscó en consecuencia caracterizar la definición de ontologías bajo una perspectiva argumentativa, con el objeto de:

- Proveer un modelo formal que posibilite estudiar distintos conceptos y metodologías vinculadas a la Web Semántica y el razonamiento argumentativo.
- Analizar las propuestas existentes para integración de modelos de razonamiento con la Web Semántica en el contexto del acercamiento propuesto a fines de mejorar su eficiencia computacional y claridad conceptual.
- Identificar nuevas líneas de investigación que puedan emerger a partir de la combinación de técnicas de representación de conocimiento en la Web y la Web Semántica junto con el razonamiento argumentativo.

Volviendo al tema de la inconsistencia en ontologías, se puede afirmar que mucha gente encuentra difícil crear y usar ontologías OWL (Wang et al., 2005). Esto ocurre porque una de las mayores dificultades consiste en "depurar" las ontologías: descubrir por qué un razonador ha inferido que una clase es "insatisfacible" (inconsistente). Aún para gente que entiende OWL y el significado de la Lógica para la Descripción subyacente, descubrir por qué los conceptos son insatisfacibles puede ser difícil. Muchos de los razonadores por *tableaux* modernos no proveen una explicación del porqué una clase es insatisfacible.

Según la definición encontrada en la literatura, una ontología es inconsistente si su interpretación es un modelo vacío (Baader et al., 2003). Una definición más operativa de la expresión "ontología inconsistente" dice que una ontología es considerada inconsistente cuando un concepto definido en la misma debería ser vacío y, sin embargo, el usuario ha declarado que la ontología tiene elementos perteneciendo a dicho concepto.

En dicho caso, hay dos maneras (no mutuamente excluyentes) de trabajar con ontologías inconsistentes (o en todo caso con bases de conocimiento inconsistentes): una manera consiste en reparar la ontología; la otra manera consiste en utilizar un método de razonamiento no-estándar para lidiar con la inconsistencia. De los dos métodos anteriores, en esta Tesis hemos decidido trabajar con el segundo (a pesar de que se discute también el primero). En consecuencia, la hipótesis de trabajo de este trabajo de Tesis puede ser expresada como la siguiente:

Hipótesis: *La argumentación rebatible permite obtener respuestas significativas a partir de una ontología inconsistente, permitiendo, además, tener una explicación del porqué se ha arribado a una conclusión determinada. Las respuestas obtenidas corresponderán básicamente a la determinación de la pertenencia de un individuo a un concepto determinado en presencia de una ontología inconsistente.*

La otra hipótesis de trabajo con la que nos manejamos es la siguiente:

Hipótesis: *La argumentación rebatible puede ser utilizada para obtener respuestas a consultas realizadas a partir de la integración de dos o más ontologías mutuamente inconsistentes.*

El resto del capítulo se halla estructurado de la siguiente forma: En la Sección 1.1 se presenta una introducción a la temática de la representación de ontologías en la Web Semántica. Seguidamente, en la Sección 1.2 se introducen los fundamentos del razonamiento en presencia de información incompleta y potencialmente contradictoria utilizando el razonamiento argumentativo. En la Sección 1.3 se introduce inicialmente la propuesta de razonamiento en la Web Semántica usando argumentación rebatible, la que se va a desarrollar en profundidad en el resto de la Tesis. Luego, en la Sección 1.4 se detallan las contribuciones presentadas en este trabajo de Tesis. Finalmente, en la Sección 1.5 se explica la organización del resto de la Tesis.

1.1. Ontologías en la Web Semántica

La *World Wide Web* ha sido posible gracias a un conjunto de protocolos que garantizan la interoperabilidad en varios niveles (Decker et al., 2000). El protocolo TCP/IP asegura el transporte de bits; HTTP y HTML aseguran una manera estándar de recuperar y presentar documentos de texto enlazados mediante hipervínculos. Las aplicaciones que fueron y son capaces de utilizar este tipo de infraestructura han producido la web que conocemos hoy en día.

Pero, además hay que notar que la web que conocemos actualmente es una web de segunda generación: la *primera generación* comenzó con páginas HTML escritas a mano; en

cambio, la *segunda generación* está compuesta por páginas HTML generadas automáticamente junto con páginas activas. Estas dos generaciones de la web fueron diseñadas para el procesamiento directo por usuarios humanos (lectura, *browsing*, completación de formularios).

Ahora, se desea avanzar un paso más. La *tercera generación* de la web es lo que se conoce como la *web del conocimiento* y tiene como objeto representar conocimiento de manera que el mismo sea procesable automáticamente por programas de computadora o *agentes*. La *Web Semántica* (Berners-Lee et al., 2001) es una visión futura de la web que implementa la web de tercera generación donde la información tiene significado exacto, permitiendo así que las computadoras entiendan y razonen en base a la información hallada allí.

Como mencionamos anteriormente, la Web actual está formada principalmente por documentos de texto. La naturaleza libre del texto hace imposible que el contenido de los documentos sea procesado en formas más complejas que las planteadas por el área de la Recuperación de Información (Frakes, 1992); es decir, básicamente el análisis consiste en analizar los términos contenidos en cada página web. Para avanzar sobre esta situación indeseable, la idea fundamental de la Web Semántica consiste en que agentes inteligentes accedan a *recursos de datos* distribuidos en Internet para lograr objetivos establecidos por los usuarios propietarios de dichos agentes. Debido a que los datos contenidos en dichos recursos deben ser comprendidos por los agentes, la semántica de los datos debe ser definida en una forma precisa que permita su procesamiento automático. Dicha semántica es especificada por medio de *metadatos* (para diferenciarlos de los datos actuales). Estos metadatos son definidos por medio de *ontologías*. En el contexto de conocimiento compartido, el término *ontología* se refiere a la especificación de una conceptualización. Esto es, una ontología es una descripción de los conceptos y relaciones que existen para un agente o una comunidad de agentes (Gruber, 1993).

Así, una *ontología* es un vocabulario controlado que describe objetos y relaciones entre ellos en una manera formal, y tiene una gramática para usar los términos del vocabulario para expresar algún concepto significativo dentro de un dominio de interés. Dicho vocabulario es utilizado para realizar aserciones y consultas. Los compromisos ontológicos son acuerdos para usar el vocabulario en una manera consistente para lograr el intercambio de conocimiento. Una *ontología formal* es un vocabulario controlado expresado en un lenguaje de representación de ontologías. En general, las ontologías describen taxonomías entre clases de individuos cuyas características son expresadas por medio de reglas.

Una situación indeseable es aquella en la que una ontología es inconsistente. Esta situación no es fortuita. Debido a que las ontologías pueden convertirse en entes complejos, un ingeniero de conocimiento que desarrolla una ontología puede incurrir en definiciones inconsistentes ya sea por errores involuntarios o porque el campo que se halla modelando es inherentemente contradictorio. Es más, como las ontologías son usualmente desarrolla-

das independientemente por autores diferentes, su combinación puede resultar en incoherencias e inconsistencias. El problema de combinar dos o más ontologías diferentes para obtener un ontología unificada y consistente es conocido como *integración de ontologías*. Esta situación se puede caracterizar como *conflictos entre reglas* (Antoniou and Bikakis, 2007), los cuales se dan en los niveles de la capa de ontologías y en la capa de lógica y razonamiento. A nivel de la capa de ontologías, tenemos: herencia *default* entre ontologías, mezcla de ontologías; y, a nivel de la capa de lógica y razonamiento, tenemos: reglas con excepciones como una manera natural de representar razonamiento con información incompleta.

En particular, en el contexto de esta Tesis, estamos interesados en ontologías expresadas en las llamadas *Lógicas para la Descripción* (o en inglés, *Description Logics*) (Baader et al., 2003), las cuales subyacen en los lenguajes de representación estándar de la Web Semántica (véase el Capítulo 2). Sintéticamente, una ontología $\Sigma = (T, A)$ consiste de dos conjuntos finitos y mutuamente disjuntos. En particular, el conjunto T (*Tbox* por *caja terminológica*) introduce la *terminología* y el conjunto A (Abox por *caja asercional*) contiene *aserciones* acerca de objetos particulares en el dominio de aplicación. Con más detalle, las sentencias de la Tbox permiten definir conceptos (o clases) y propiedades (o roles) y son básicamente de la forma $C \sqsubseteq D$ (*inclusiones*) y $C \equiv D$ (*igualdades*), donde C y D son descripciones de conceptos (posiblemente complejas).

Como veremos en los capítulos posteriores, es posible dar una semántica a las ontologías al considerar la correspondencia existente entre las Lógicas para la Descripción y la Lógica de Primer Orden. Básicamente, los nombres de conceptos se corresponden con predicados unarios mientras que los roles se corresponden con predicados binarios. Así, las expresiones de la Tbox pueden ser correspondidas con expresiones de la Lógica de Primer Orden y las aserciones de la Abox con un conjunto de hechos.

La tarea de razonamiento más importante en las Lógicas para la Descripción es el testeo de pertenencia de un individuo a a una clase C con respecto a una ontología Σ. La resolución de este problema consiste de hallar una interpretación que sea modelo al mismo tiempo de Σ y de $C(a)$. Un caso especial se da cuando una ontología es inconsistente. Lógicamente, esta situación está caracterizada porque todos los modelos de la ontología en cuestión son vacíos. En este caso no es posible dar una solución a una tal consulta en el marco del tratamiento tradicional de las Lógicas para la Descripción (en esta Tesis, cuando nos refiramos al *tratamiento tradicional de ontologías*, lo haremos en el sentido de (Baader et al., 2003)). En esta Tesis, la solución a este problema la hallaremos en el marco de la utilización de un formalismo de razonamiento no-monótono llamado *argumentación rebatible* aplicado a las ontologías expresadas en Lógicas para la Descripción.

1.2. Razonamiento con argumentación rebatible

En la matemática el conocimiento se representa con lógica de primer orden (FOL), la cual es un poderoso formalismo de representación de conocimiento. Sin embargo, la FOL tiene severas restricciones para las aplicaciones de Inteligencia Artificial. Estas restricciones incluyen la monotonicidad y su limitada habilidad para representar información incierta e incompleta. Además, se presentan problemas cuando la información que se posee es contradictoria.

La argumentación se preocupa de resolver los problemas anteriores. La argumentación surgió en el campo de la filosofía. La argumentación evolucionó desde un enfoque informal hacia enfoques formales. Los sistemas formales de argumentación se caracterizan por representar los mismos elementos que los sistemas informales usando lenguajes formales y aplicando reglas de inferencias formales en ellos (Carbogim et al., 2000). Carbogim et al. (2000) comentan que:

> *"los argumentos tienen la forma de la demostración lógica pero no su fuerza".*

Es decir, los argumentos tienen el mismo aspecto que las demostraciones lógicas pero sus conclusiones no son necesariamente verdades en el sentido de la lógica.

La argumentación provee una perspectiva diferente al razonamiento no-monótono y al razonamiento rebatible, en el cual una afirmación es aceptada o rechazada en la base de los argumentos en favor o en contra de ella, y en el hecho de si dichos argumentos son atacados y derrotados por otros (Carbogim et al., 2000). Esta visión es soportada por la *argumentación rebatible*.

Los marcos de argumentación tienen generalmente los siguientes elementos (Chesñevar, Maguitman and Loui, 2000; Prakken and Vreeswijk, 2002; Bench-Capon and Dunne, 2007): un lenguaje lógico subyacente, un concepto de argumento, un concepto de conflicto entre argumentos, una noción de derrota entre argumentos y otra noción de cuándo un argumento es aceptable. A continuación, describiremos someramente a cada uno de los elementos de un sistema argumentativo.

Lenguaje lógico subyacente: El lenguaje lógico subyacente será el de la lógica formal de primer orden donde están definidas una o más relaciones de consecuencia monótonas que establecen la base para derivar argumentos. En general, muchos sistemas de argumentación involucran una base de conocimiento $\mathcal{K} = (\Pi, \Delta)$ que provee el conocimiento del dominio para un agente formalizado en un lenguaje de primer orden $\mathcal{L}$. Este conocimiento de dominio involucra usualmente un conjunto Π de reglas estrictas y hechos y un conjunto Δ de reglas rebatibles.

Argumento: Un argumento es una prueba rebatible obtenida a partir de la base de conocimiento $\mathcal{K}$ por medio de la aplicación de convenientes reglas de inferencia (quizá rebatibles) asociadas con el lenguaje lógico subyacente $\mathcal{L}$.

Conflicto entre argumentos: Dados dos argumentos $\mathcal{A}$ y $\mathcal{B}$, el *conflicto* (o *ataque*) entre argumentos surge siempre que $\mathcal{A}$ y $\mathcal{B}$ no puedan ser aceptados simultáneamente (típicamente debido a algún tipo de contradicción lógica).

Derrota entre argumentos: Como el lenguaje lógico subyacente es monótono, el agregado de nueva información no invalida argumentos existentes ni conclusiones derivadas previamente; por lo tanto, en una base de conocimiento pueden coexistir argumentos en conflicto. El carácter no-monótono de la argumentación surge del hecho de que algunos argumentos serán preferidos sobre otros y el formalismo debería tener medios para decidir cuál de dichos argumentos es aceptable.

La noción de derrota está usualmente basada en alguna medida comparativa para los argumentos y se utiliza algún criterio basado en esta medida para decidir la identidad del argumento ganador. Una manera es asignar prioridad a las reglas en un sistema basado en reglas, otra forma son los criterios de especificidad y el de directitud.

Formalmente, muchos sistemas de argumentación proveen un criterio de preferencia que define un orden parcial entre argumentos, permitiendo determinar cuando $\mathcal{A}$ debe preferirse sobre $\mathcal{B}$. Esto define una relación de *derrota* (*defeats*). Dado el conjunto $Args$ de argumentos obtenidos de una base de conocimiento $\mathcal{K}$, vale que la relación de ataque $attacks \subseteq Args \times Args$. Cuando el argumento $\mathcal{A}$ se prefiere sobre un argumento $\mathcal{B}$, se dice que $\mathcal{A}$ derrota a $\mathcal{B}$. Además, vale que $defeats \subseteq attacks$.

Argumento aceptable: El objetivo de un sistema argumentativo es el de determinar qué afirmaciones y qué argumentos son aceptables. La noción de aceptabilidad varía de formalismo en formalismo, pero intuitivamente la noción se corresponde a que un argumento no será aceptable si es derrotado por algún argumento aunque sea capaz de derrotar a un argumento en conflicto con él. Para determinar si un argumento dado $\mathcal{A}$ es *aceptable* (o *triunfador* o *justificado*), se lleva a cabo un proceso dialéctico, en el cual se tienen en cuenta derrotadores para $\mathcal{A}$, derrotadores para estos derrotadores, y así sucesivamente. Por ello, para este proceso se deben tener en cuenta a todos los argumentos a favor y en contra de una afirmación dada antes de tomar una decisión.

1.3. Razonamiento en la Web Semántica usando argumentación rebatible

Como hemos explicado anteriormente, la idea subyacente del programa de la Web Semántica es permitir que los agentes razonen sobre la información en la Web para alcanzar objetivos establecidos por sus usuarios propietarios. Para lograr una interacción

efectiva el conocimiento de los agentes debe ser compartido efectivamente. Para lograr este objetivo, la información en la web debe tener una semántica o significado precisos. Dicha semántica es formulada por medio de definiciones de ontologías. Estas ontologías son procesadas por los agentes para entender los datos del dominio de aplicación.

En un sistema abierto como la Web, se da el caso en el que muchos agentes entran y salen todo el tiempo y deben interactuar entre sí, para poder lograr sus objetivos. Un caso típico de interacción se da cuando un agente requiere servicios de otro agente mediante protocolos de comunicación previamente establecidos. Las ontologías complementan estos protocolos de interacción, pues un agente se compromete con una ontología, determinando la forma en que este agente ve al mundo. Así, la forma en que el agente ve al mundo impacta en el vocabulario que utiliza. En este contexto un agente Ag_i será descripto por la ontología Σ_i con la que se compromete y un sistema multiagente será simplemente un conjunto de agentes $A = \{Ag_1, \ldots, Ag_n\}$ (Laera et al., 2006).

Una ontología Σ puede ser definida como una tupla $(C, \leq_C, R, \leq_R, I)$, donde C y R son dos conjuntos disjuntos y sus elementos son llamados *conceptos* y *roles*. Las relaciones $\leq_C$ y $\leq_R$ son órdenes parciales sobre C y R respectivamente y son llamados la *jerarquía de conceptos* y la *jerarquía de roles* respectivamente. Los elementos de I pueden ser instancias de conceptos en C y pueden estar interconectados con otros elementos en I por una relación en R. Tales elementos son conocidos como *individuos*.

Como cada agente se compromete con una ontología que define su vocabulario, para lograr la interacción entre dos agentes, éstos deben *alinear* sus respectivas ontologías. Una *alineación* consiste de un conjunto de correspondencias entre las dos ontologías. Abstractamente, una correspondencia entre una ontología Σ_1 y otra ontología Σ_2 puede ser descripta como una tupla $m = (e, e', R)$ donde e es un objeto (concepto, rol o individuo) de Σ_1, e' es un objeto de Σ_2 y R es una relación (*e.g.*, subsume, es-subsumido, equivalente, etc).

Un problema que surge entonces es que los lenguajes actuales permiten representar información incompleta e inherentemente contradictoria pero los razonadores actuales son incapaces de gestionar tales definiciones en algunos casos. Así, la argumentación rebatible se presenta como una opción atractiva para modelar dominios de aplicación con información incompleta y potencialmente contradictoria. Por otro lado, el campo de la argumentación rebatible se halla lo suficientemente maduro como para proveer implementaciones de razonadores sobre bases de conocimiento dialéctico. Dichas implementaciones pueden ser utilizadas como la base sobre la cual implementar razonadores capaces de inferir conocimiento tentativo en la base de ontologías contradictorias. Habiendo explicado el contexto en que la investigación llevada a cabo en esta Tesis se solapa con la teoría de agentes en el marco de la Web Semántica, en la siguiente sección abordamos con más detalle la solución propuesta en esta Tesis para el tratamiento de ontologías inconsistentes en el marco de la iniciativa de la Web Semántica.

1.3.1. Visión general de la solución propuesta

En este trabajo de tesis presentaremos un marco que permite razonar con ontologías inconsistentes en el contexto de la Web Semántica. Asumiremos, de acuerdo al estándar propuesto por la W3C, que las ontologías en la Web Semántica se representan en un lenguaje de representación de ontologías como OWL (por *Ontology Web Language*) (véase la Sección 2.2.5).

El significado de las ontologías representadas en OWL puede ser dado mediante una semántica basada en Lógicas para la Descripción. Cuando se utiliza una semántica basada en Lógicas para la Descripción, se asume un esquema de trabajo como el presentado en la Figura 1.1. Una base de conocimiento $\mathcal{KB}$ es representada por medio de una ontología Σ y la misma es descripta por medio de un lenguaje de descripción de ontologías basado en Lógicas para la Descripción. Una ontología Σ está compuesta por una caja terminológica (o Tbox) T y una caja asercional (o Abox) A. La Tbox define el conjunto de conceptos, atributos y relaciones entre conceptos que valen en la ontología. La Abox define el conjunto de individuos y las relaciones que se cumplen entre ellos. Una de las tareas de razonamiento sobre ontologías consiste en determinar si un dado individuo pertenece a un concepto determinado. Dicha tarea de razonamiento es llevada a cabo por medio de un motor de razonamiento ontológico. Así, en este marco de trabajo, los agentes, representados por programas de aplicación, realizan consultas Q respecto de la pertenencia de un individuo a a un concepto C. Esta consulta es procesada por el motor de razonamiento ontológico, el cual genera una respuesta.

En cambio, consideremos un marco de razonamiento argumentativo como el que se presenta en la Figura 1.2. En este caso, el conocimiento se representa por medio del lenguaje de la programación en lógica extendida. Una base de conocimiento se halla representada mediante dos conjuntos: un conjunto de reglas estrictas Π y un conjunto de reglas rebatibles Δ. Tal separación se realiza de tal manera que la información conflictiva se representa en el conjunto Δ. Así, los agentes, representados por programas de aplicación, hacen consultas respecto del estado de garantía de un literal dado. El razonamiento en este contexto es realizado por un motor de inferencia argumentativo, el cual determina todas las razones a favor y en contra de tal estado de garantía.

Una ontología es inconsistente cuando todos sus modelos son vacíos. En presencia de ontologías inconsistentes, los razonadores ontológicos no son capaces de extraer conclusiones significativas. Sin embargo, a partir del análisis realizado más arriba, vemos que existen varios paralelos entre los dos acercamientos. Explicaremos estos paralelos y veremos cómo los mismos serán utilizados en esta Tesis para resolver el problema del razonamiento en presencia de ontologías inconsistentes.

Así, una base de conocimiento (u ontología) expresada en Lógicas para la Descripción puede ser expresada como una base de conocimiento (o programa rebatible) argumentativo. En particular, una Tbox puede ser expresada como un conjunto de reglas estrictas

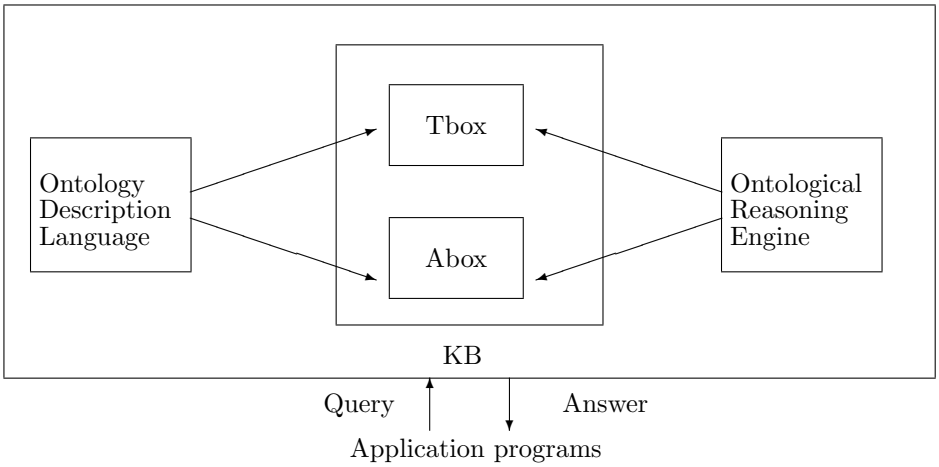


Figura 1.1: Arquitectura de un sistema de representación de conocimiento basado en Lógicas para la Descripción (adaptado de (Baader et al., 2003, Fig. 2.1))

y rebatibles, mientras que una Abox puede ser expresada como un conjunto de hechos. De esta manera, una consulta de pertenencia de un individuo a a un concepto C con respecto a una ontología inconsistente Σ será expresada como la consulta para determinar si el literal $C(a)$ se halla garantizado con respecto al programa rebatible $\mathcal{P}_\Sigma$ que se obtiene a partir de la ontología Σ. En esta Tesis, estudiaremos bajo qué condiciones tal acercamiento se puede llevar a cabo y las propiedades que se desprenden de tal enfoque.

El marco de razonamiento presentado en esta Tesis puede ser integrado a un sistema multiagente implementado con las tecnologías actuales en el contexto de la Web Semántica (véase la Figura 1.3). Básicamente, tenemos un agente mediador cuyo objetivo es permitir contestar consultas Q con respecto a una ontología Σ_{Owl} y una base de datos Db. La ontología Σ_{Owl} define el significado de los datos mientras que en la bases de datos Db se hallan almacenados los datos actuales. [1] El agente recupera la ontología Σ_{Owl} de un servidor de ontologías y la base de datos de un servidor de bases de datos. El usuario especifica las consultas a través de una interfaz web y obtiene los resultados a través de la misma. El agente también analiza la ontología. Para esto es necesario determinar si la misma es o no inconsistente (quizá con la ayuda de otros agentes en un sistema multiagente). Entonces, la ontología Σ_{Owl} se interpreta como una base de conocimiento $\mathcal{P} = (\Pi, \Delta)$ en un lenguaje argumentativo y se utiliza un motor de inferencia argumentativo para resolver consultas de pertenencia de individuos a conceptos en el contexto de una ontología inconsistente.

[1] En este contexto pensamos en una base de datos RDF donde los registros se pueden corresponder con la Abox de una ontología.

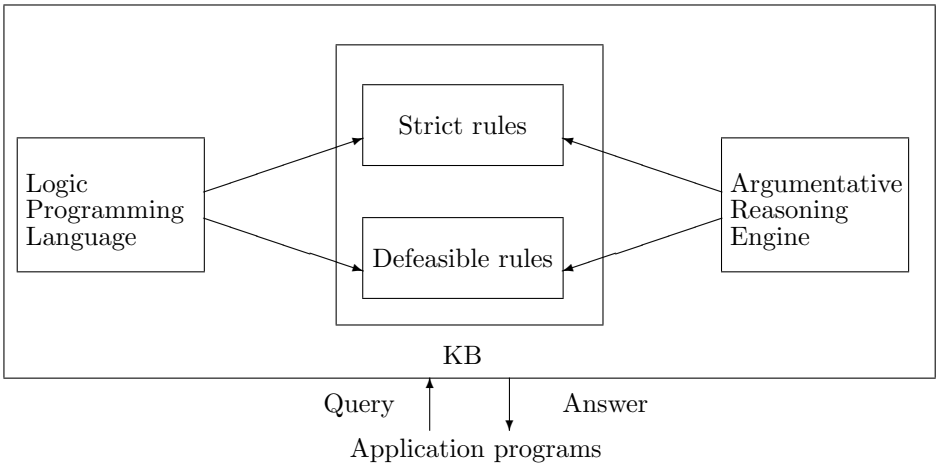


Figura 1.2: Arquitectura de un sistema de representación de conocimiento basado en argumentación rebatible

1.4. Contribuciones de esta Tesis

Las contribuciones principales de esta Tesis pueden resumirse como se detalla a continuación:

- Se presenta un resumen de los lenguajes de representación de conocimiento para la Web y la Web semántica junto con las Lógicas para la Descripción que subyacen los mismos. El objetivo es dar una visión global de la iniciativa de la Web Semántica y cómo los elementos presentados para la definición de ontologías se integran con la solución propuesta en esta Tesis para manipular ontologías potencialmente inconsistentes.

  Tal tarea se lleva a cabo en el Capítulo 2.

- Se presenta un resumen del marco de razonamiento con la Programación en Lógica Rebatible. Este acercamiento complementa aquel presentado en la Tesis de Magíster (Gómez, 2003) y tiene el objetivo de brindar un marco autocontenido mediante el cual comprender la solución mostrada en esta Tesis para razonar con ontologías posiblemente inconsistentes en el marco de la aplicación de la Programación en Lógica Rebatible.

  Tal tarea se lleva a cabo en el Capítulo 3.

- Se presenta un formalismo para expresar ontologías expresadas en Lógicas para la Descripción en un marco argumentativo. La capacidad de razonar con ontologías inconsistentes se ve mejorada ya que el formalismo utilizado no sólo permite extraer conclusiones en presencia de ontologías inconsistentes sino además ser capaces de

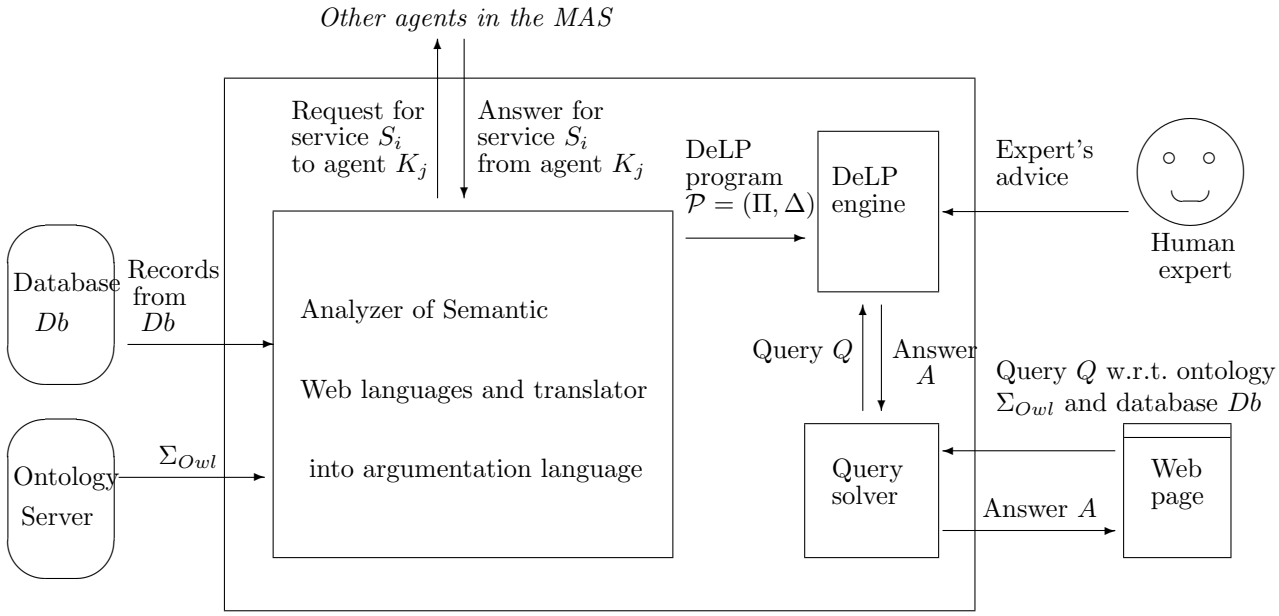


Figura 1.3: Un esquema para manejar definiciones de ontologías inconsistentes con argumentación rebatible en la Web Semántica

explicar por qué se llega a una conclusión determinada. Se estudian las posibilidades de representación de los ejemplos relevantes de la literatura en el formalismo de representación de ontologías presentado. Se estudia las aplicaciones de la propuesta para la solución al problema de la integración de ontologías. También se muestra cómo la propuesta presentada puede modificarse para comportarse como algunas de las propuestas analizadas en el trabajo relacionado. Se estudia también una aplicación del formalismo presentado a la integración de ontologías.

Tal tarea se lleva a cabo en el Capítulo 4 y complementa el trabajo realizado en (Gómez et al., 2006*a*, 2007*b*, 2008*a*).

- Se estudian las propiedades emergentes de la propuesta para razonar con ontologías posiblemente inconsistentes utilizando la Programación en Lógica Rebatible.

  Se muestra que, bajo la hipótesis de que la ontología en cuestión sea representable en la Programación en Lógica, y cuando la misma es consistente, las conclusiones obtenidas por el marco propuesto en esta Tesis están contenidas en el conjunto de conclusiones que obtiene un razonador tradicional. Pero, en el caso en que la ontología es inconsistente, el marco propuesto en esta Tesis es capaz de obtener respuestas y, además, explicar cómo las ha obtenido, aún cuando los razonadores tradicionales no son capaces de generar ninguna (salvo la detección de la inconsistencia). Otros resultados muestran que la propuesta presentada satisface los postulados de evaluación planteados por autores especializados en la literatura del campo (como (Huang

et al., 2005)). Se discute además cómo extender la propuesta presentada al problema de la mezcla de ontologías.

Se presenta en el Capítulo 5 y complementa el trabajo realizado en (Gómez et al., 2007*a*, 2008*a*).

- Se presenta una aplicación del formalismo presentado a la web y, en particular, a la tecnología de formularios web. La propuesta presenta entonces una aplicación del razonamiento sobre ontologías inconsistentes por medio de la argumentación rebatible al modelado de formularios en la World Wide Web. Para el diseñador del formulario, el significado pretendido de los valores de los campos de un formulario frecuentemente involucra otras características que no son restricciones por sí mismas, sino más bien *atributos* emergentes del formulario, los cuales proveen conclusiones plausibles en el contexto de información incompleta y potencialmente contradictoria. Como el valor de tales atributos puede cambiar en presencia de nuevo conocimiento, los llamamos *atributos rebatibles*.

  Proponemos extender los formularios web para incorporar atributos rebatibles como parte del conocimiento que puede ser codificado por el diseñador del formulario. Veremos cómo dicho conocimiento puede ser especificado mediante un programa DeLP, y posteriormente, como una ontología expresada en Lógicas para la Descripción. Así, la extensión propuesta permite la especificación de guiones para razonar acerca de los campos del formulario utilizando una base de conocimiento rebatible, expresada en términos de un Programa Lógico Rebatible.

  Se presenta en el Capítulo 6 y complementa el trabajo realizado en (Gómez et al., 2005*c*, 2008*b*,*d*).

1.4.1. Trabajos previos relevantes

A continuación, se mencionan los principales trabajos realizados en los últimos años durante el desarrollo de esta Tesis. Muchos de sus contenidos se hallan integrados a lo largo de los distintos capítulos de la misma. Para cada uno de ellos se detalla brevemente su contribución. Además, los trabajos son presentados en el orden cronológico en el que fueron concebidos.

1. En el trabajo *"Embedding Defeasible Argumentation in the Semantic Web: an ontology-based approach"* (Gómez et al., 2005*b*) se presenta un primer acercamiento para modelar ontologías mediante programas rebatibles. También se presenta un esbozo de un álgebra para modelar la dinámica de cambio de programas rebatibles cuya semántica se halla basada en Programación en Lógica Rebatible y en la teoría de Revisión de Creencias.

2. En el trabajo *"Incorporating Defeasible Knowledge and Argumentative Reasoning in Web-based Forms"* (Gómez et al., 2005*c*) se presenta un acercamiento novedoso para combinar argumentación rebatible y formularios web. Se introducen los *δ-forms* que poseen atributos rebatibles. En el ejemplo presentado, un ingeniero de conocimiento puede especificar un conjunto de criterios para establecer cuándo un cliente de un banco es candidato a la obtención de un préstamo. Debido a que los criterios mencionados pueden ser inconsistentes a partir de la información factual obtenida desde el formulario y otras fuentes de datos en la web, se muestra cómo realizar un análisis dialéctico para determinar qué atributos emergentes del formulario se hayan garantizados. Se presenta también por primera vez la noción de *redefinición de un predicado en un programa rebatible*. Esta noción permite actualizar la base de conocimiento en presencia de nueva información no sólo factualmente sino a nivel de regla.

3. En el trabajo *"A First Approach to Combining Ontologies and Defeasible Argumentation for the Semantic Web"* (Gómez et al., 2005*a*) se presenta con detalle la definición de un lenguaje para intercambio de ontologías basadas en Programas Lógicos Rebatibles en el contexto de la Web. El lenguaje presentado está basado en el Lenguaje de Marcado Extensible (XML).

4. En el trabajo *"Problems and Challenges for Ontology Integration in the Semantic Web"* (Gómez et al., 2006*b*) se presenta brevemente la problemática de la integración de ontologías en el contexto de la Web Semántica, se resumen los lenguajes de representación de ontologías en la Web Semántica, se compendian brevemente las definiciones pertinentes al campo de la integración de ontologías, y se recopilan los acercamientos al problema encontrados en la literatura del campo.

5. En el trabajo *"An Approach to Handling Inconsistent Ontology Definitions based on the Translation of Description Logics into Defeasible Logic Programming"* (Gómez et al., 2006*a*) se presenta un enfoque novedoso para tratar ontologías definidas en Lógicas para la Descripción utilizando Programación en Lógica Rebatible. Mostramos cómo obtener un programa rebatible a partir de una ontología y cómo obtener conclusiones garantizadas cuando éstas no se pueden realizar en el marco de las Lógicas para la Descripción al considerar la ontología original.

6. En el trabajo *"Inconsistent Ontology Handling by Translating Description Logics into Defeasible Logic Programming"* (Gómez et al., 2007*b*) se presenta una versión extendida del trabajo propio (Gómez et al., 2006*a*). La extensión presentada consiste en la análisis de algunas propiedades emergentes de la propuesta anterior respecto de la consistencia y completitud del acercamiento al razonamiento con ontologías

basado en la Programación en Lógica Rebatible con respecto al mismo realizado en el acercamiento tradicional de las Lógicas para la Descripción.

7. En el trabajo *"Hacia una Integración de Argumentación Rebatible y Ontologías en la Web Semántica"* (Gómez et al., 2007*a*) se presenta un acercamiento a la aplicación de un lenguaje de representación de bases de conocimiento rebatible como lenguaje de representación de ontologías. Dicho acercamiento provee mayor flexibilidad a las limitaciones impuestas por los lenguajes estándar de representación de ontologías en términos de representatividad y soporte para razonar frente a inconsistencias en el marco de la Web Semántica.

8. En el trabajo titulado *"Defeasible Reasoning in Web-based Forms Through Argumentation"* (Gómez et al., 2008*b*) se depuran los resultados hallados en los trabajos propios (Gómez et al., 2005*c*) y (Gómez et al., 2005*a*) para combinarlos en un marco unificado.

9. En el trabajo *"An Argumentative Approach to Reasoning with Inconsistent Ontologies"* (Gómez et al., 2008*a*) se presentan las δ-ontologías como un subconjunto de las Lógicas para la Descripción, que pueden contener inconsistencias y pueden ser interpretadas como programas lógicos rebatibles. Se da un conjunto de definiciones para caracterizar las tareas de recuperación en δ-ontologías y se las compara a aquellas asociadas a las ontologías expresadas en las Lógicas para la Descripción. También se aplica el marco presentado a la integración de ontologías con el acercamiento *global-as-view*.

10. En el trabajo *"Integration of Web-based Forms with Ontologies in the Semantic Web"* (Gómez et al., 2008*d*) se presenta una adaptación del concepto de δ-formulario en donde la especificación de los argumentos rebatibles se lleva a cabo mediante una ontología expresada en Lógicas para la Descripción, que puede ser inconsistente.

1.5. Organización de esta Tesis

Esta Tesis ha sido iniciada en el marco de una Beca de Iniciación a la Investigación otorgada por la Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT), comprendida entre abril de 2004 y marzo de 2007. Se incluyen todos los conceptos necesarios para que su lectura sea autocontenida, asumiéndose que se poseen conocimientos a nivel de pregrado de lógica de predicados e Inteligencia Artificial.

A continuación, se describe sintéticamente la estructura de la Tesis, especificándose los principales aportes presentados en los diferentes capítulos:

Capítulo 1: Se introducen los objetivos y el marco en el que se ha desarrollado esta investigación. Se resumen las principales contribuciones presentadas en la Tesis y se reseñan brevemente los resultados obtenidos en trabajos de investigación realizados durante su desarrollo, los que han sido publicados en diversos congresos y revistas nacionales e internacionales.

Capítulo 2: Se presentan los contenidos básicos para comprender el área de investigación de la Web Semántica. Se presentan los lenguajes de representación de conocimiento en la Web Semántica, los cuales son utilizados para representar ontologías; en particular, se explican las Lógicas para la Descripción, que subyacen a los lenguajes de representación de ontologías propuestos en el último estándar del Consorcio de la World Wide Web. Se presentan también la problemática de la representación de conocimiento inconsistente y la integración de ontologías.

Capítulo 3: Se presenta el formalismo de la Programación en Lógica Rebatible como un medio para modelar el razonamiento argumentativo en presencia de información incompleta y potencialmente contradictoria.

Capítulo 4: Se introducen las δ-ontologías como un formalismo de representación de ontologías potencialmente contradictorias. Las δ-ontologías son básicamente ontologías expresadas en las Lógicas para la Descripción pero que son interpretadas como programas lógicos rebatibles, lo cual permite deducir conclusiones en presencia de ontologías inconsistentes. Además, se introduce una aplicación a la integración de ontologías.

Capítulo 5: Se estudian las propiedades emergentes de la propuesta para razonar con ontologías posiblemente inconsistentes utilizando la Programación en Lógica Rebatible. Se muestra que, bajo la hipótesis de que la ontología en cuestión sea representable en la Programación en Lógica, y cuando la misma es consistente, las conclusiones obtenidas por el marco propuesto en esta Tesis están contenidas en el conjunto de conclusiones que obtiene un razonador tradicional. Pero, en el caso en que la ontología es inconsistente, el marco propuesto en esta Tesis es capaz de obtener respuestas y, además, explicar cómo las ha obtenido. Otros resultados muestran que la propuesta presentada satisface los postulados de evaluación planteados por autores especializados en la literatura del campo.

Capítulo 6: Se introducen los δ-formularios como una extensión de los formularios Web tradicionales. Los δ-formularios son extendidos con ontologías expresadas como programas lógicos rebatibles lo cual permite inferir el valor de atributos rebatibles expresados en términos de los valores de los campos del formulario e interpretados

de acuerdo a la especificación dada por la ontología que subyace el contenido del formulario.

Capítulo 7: Se presentan las principales conclusiones obtenidas en la Tesis y se vislumbran futuras líneas de investigación para extender los resultados hallados en la misma.

Bibliografía: Lista de referencias bibliográficas utilizadas en la elaboración de la Tesis, junto con un breve comentario personal sobre su contenido.

1.6. Resumen

En este capítulo introductorio se presentaron las motivaciones subyaciendo este trabajo de Tesis. Primeramente se introdujo el marco de la iniciativa de la Web Semántica y cómo los agentes inteligentes podrán navegar fuentes de información en la misma. Para tal fin, se utilizan ontologías para modelar el significado de los datos de tales fuentes de información. Debido a que las ontologías pueden contener inconsistencias, se presentaron los principales elementos del marco de la argumentación rebatible en el marco del razonamiento no-monótono, el cual permite razonar en presencia de bases de conocimiento incompletas y potencialmente inconsistentes. Luego, se presentó el esquema de la solución propuesta en esta Tesis al razonamiento e integración de ontologías inconsistentes en el marco de la Web Semántica. Además, se mencionaron las contribuciones principales realizadas en el contexto de este trabajo de Tesis junto con las publicaciones realizadas durante los últimos años, las cuales sirvieron de sustrato para elaborar esta Tesis. Finalmente, se presentó la organización del resto de esta Tesis.

Capítulo 2

Ontologías y razonamiento en la Web

La arquitectura de la Web Semántica ha sido desarrollada como un conjunto de lenguajes estandarizados por el Consorcio de la World Wide Web. Las Lógicas para la Descripción son muy importantes en este esquema ya que proveen el fundamento semántico de uno de los lenguajes llamado OWL. El lenguaje OWL permite definir ontologías, que son especificaciones de una conceptualización. Tales ontologías serán muy importantes en la Web Semántica ya que definirán con exactitud el significado de los recursos, permitiendo que agentes inteligentes con poder delegado por sus usuarios sean capaces de interpretar el conocimiento allí presente.

En este capítulo presentamos los fundamentos de la Web Semántica y el razonamiento con ontologías, los cuales incluyen a los lenguajes de descripción de ontologías, los formalismos lógicos subyacentes así como temas relacionados como la integración y mezcla de ontologías. Parte de este capítulo está basado y extiende las ideas presentadas en el artículo (Gómez et al., 2006*b*) y se estructura de la siguiente manera: En la Sección 2.1 se presentan los fundamentos de la Web Semántica. Posteriormente, en la Sección 2.2 se presenta una breve síntesis de los lenguajes utilizados para representar conocimiento en la Web. Luego, en la Sección 2.3 se introduce el formalismo de las Lógicas para la Descripción. A continuación, en la Sección 2.4 se discute la problemática de las inconsistencias en las definiciones de ontologías. Finalmente, en la Sección 2.5, se discute el problema de la integración de ontologías.

2.1. Una nueva web: La Web Semántica

2.1.1. Motivaciones

La *World Wide Web* (WWW) actual está basada principalmente en documentos escritos para su presentación visual para usuarios humanos. Sin embargo, para obtener todo el potencial de la Web es necesario que los programas de computadoras o *agentes* sean capaces de comprender la información en la misma.

La *Web Semántica* (Berners-Lee et al., 2001) es una visión futura de la Web donde la información tiene significado exacto, permitiendo así que las computadoras entiendan y razonen en base a la información hallada en la Web. La Web Semántica propone resolver el problema de la asignación de semántica a los recursos por medio de *metadatos* cuyo significado es dado a través de *definiciones de ontologías*.

En el contexto del área de compartir conocimiento, el término *ontología* se refiere a la especificación de una conceptualización. Esto es, una ontología es una descripción de los conceptos y relaciones que pueden existir para un agente o una comunidad de agentes (Gruber, 1993). Se tiene interés en que un agente inteligente, quizá inmerso en un *sistema multiagente*, pueda razonar a partir de esas ontologías y los datos que son descriptos por ellas, obteniendo así conclusiones *racionalmente justificadas* a partir de dichos datos. Por lo tanto, el concepto más importante de la Web Semántica es que los documentos son entendibles por las máquinas y no solamente por los usuarios. En consecuencia, agentes inteligentes con poder delegado de los usuarios pueden acceder a los datos y razonar acerca de su contenido para lograr los objetivos propuestos.

Entre las interpretaciones para la Web Semántica (van Harmelen, 2006), podemos hallar las siguientes: (i) la Web Semántica es una biblioteca universal para acceso humano, y, (ii) la Web Semántica es el hábitat para agentes automatizados y servicios web.

En la primera interpretación, el objetivo principal de la Web Semántica es permitir la integración de fuentes de datos estructuradas y semi-estructuradas sobre la Web. La receta principal consiste en exponer las fuentes de datos de la Web en formato RDF,[1] y utilizar RDF-Schema para expresar la semántica pretendida de dichas fuentes de datos, con el objetivo de permitir la integración e inesperado reuso de las mismas.[2] Un caso típico de uso para esta versión de la Web Semántica es la combinación de datos geográficos con un conjunto de calificaciones de consumidores para restaurantes de tal manera de proveer una fuente de información enriquecida.

En la segunda interpretación, el objetivo de la Web Semántica es mejorar la World Wide Web actual. Los casos de uso típicos son motores de búsqueda mejorados, personalización dinámica de sitios web, y enriquecimiento semántico de páginas web existentes. La fuente de los metadatos requeridos en esta versión de la Web Semántica provienen de fuentes de extracción automática: extracción de conceptos, reconocimiento de entidades nombradas, clasificación automática, etc.

La comunidad científica ya ha encarado con un cierto éxito la solución de los objetivos del programa de la Web Semántica. Sin embargo, pensamos que todavía hay problemas sin resolver. En particular, en los enfoques actuales, las ontologías son descriptas en un

[1] Los lenguajes RDF y RDF-Schema se introducen más adelante en la Sección 2.2.

[2] Para relacionar estas afirmaciones con la teoría de bases de datos, podemos pensar en el siguiente ejemplo. La información expresada en RDF-Schema sería equivalente al esquema de una base de datos mientras que las tuplas en RDF serían equivalentes a los registros de las tablas en una base de datos.

lenguaje de representación de ontologías; dando más detalles, el *Lenguaje de Ontologías Web* (OWL) (McGuiness and van Harmelen, 2004) es el lenguaje propuesto por el estándar del Consorcio de la W3C para tal fin. Aunque dicho lenguaje provee un número expresivo de constructores, el mismo adolece del problema de no permitir razonar en presencia de información *incompleta* y potencialmente *contradictoria*.

2.1.2. Bases de conocimiento en la web: Ontologías

El centro de todas las aplicaciones de la Web Semántica es el uso de ontologías. Las ontologías usualmente son definidas como (Gruber, 1993):

> *Una ontología es una especificación formal y explícita de la conceptualización de un dominio de interés.*

En dicha definición se pone énfasis en que la conceptualización es formal y por lo tanto permite su procesamiento por medio de una computadora. Las ontologías consisten de conceptos (o clases), relaciones (o propiedades), instancias y axiomas. Por lo tanto, usualmente se suele definir una ontología como una tupla (C, R, I, A), donde C es un conjunto de conceptos, R es un conjunto de relaciones, I un conjunto de instancias y A un conjunto de axiomas.

Los lenguajes de representación de ontologías en la Web Semántica han evolucionado en la última década. La web original es descripta con HTML (*HyperText Markup Language*). Los datos de bases de datos pueden ser serializados en XML (*eXtensible Markup Language*). Inicialmente se comenzó con RDF (*Resource Description Framework*), el cual tiene una semántica basada en grafos pero que usualmente se serializa como triplas en el lenguaje XML. Luego, para la definición de clases se utilizó RDF Schema. Luego surgieron DAML en la comunidad europea y OIL en la comunidad norteamericana; de la unión de ambos apareció DAML+OIL. Finalmente, OWL (*Ontology Web Language*) es el último estándar propuesto. En la siguiente sección exploramos con un poco más de precisión los fundamentos de estos lenguajes de representación de conocimiento en la Web Semántica.

2.2. Lenguajes de representación de conocimiento en la Web Semántica

Los principios de las Web Semántica son implementados en las capas de las tecnologías y estándares Web (Koivunen and Miller, 2001). Las capas son presentadas en la Figura 2.1. Las capas Unicode y URI hacen que sea posible utilizar conjuntos de caracteres internacionales y brindan medios para la identificación de objetos en la Web Semántica. La capa XML con espacios de nombres y definiciones de esquemas hacen que sea posible integrar definiciones en la Web Semántica con otros estándares basados en XML. Con

RDF y RDFSchema es posible construir sentencias acerca de objetos con URI's y definir vocabularios que pueden ser referidos por medio de URI's. Esta es la capa donde es posible dar tipos a los recursos y enlaces. La capa de Ontologías (en inglés, *Ontology vocabulary*) soporta la evolución de vocabularios ya que en ella se pueden definir relaciones entre los diferentes conceptos. Con la capa de la Firma Digital (en inglés, *Digital Signature*) para la detección de alteraciones a documentos, estas son las capas que están siendo actualmente estandarizadas por los grupos de trabajo de la W3C. Las capas del tope —Lógica, Prueba y Confianza (en inglés, *Logic*, *Proof*, y *Trust*, respectivamente)— están siendo investigadas actualmente. La capa Lógica permite la escritura de reglas mientras que la capa de Prueba ejecuta las reglas junto con el mecanismo de Confianza para aplicaciones donde se debe confiar en la veracidad de la prueba.

A continuación presentamos resumidamente los conceptos fundamentales de los lenguajes de representación de conocimiento en la Web Semántica.

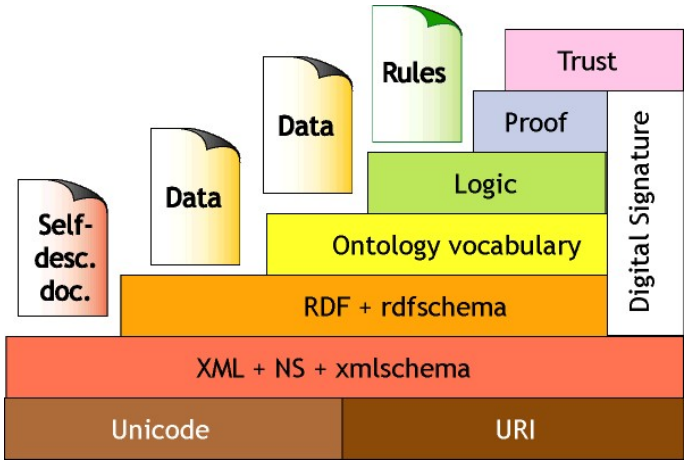


Figura 2.1: Capas de la Web Semántica

2.2.1. Hypertext Markup Language

El *Lenguaje de Marcado de Hipertextos* (en inglés, *HyperText Markup Language* o HTML) (Raggett et al., 1999) es un lenguaje de marcado diseñado para la creación de páginas web que contengan hipertexto y otra información para ser mostradas en un navegador (en inglés, *browser*). HTML es usado para estructurar la información (denotando cierto texto como encabezados, párrafos, listas, etc.) y para describir la apariencia de un documento.

Las *Hojas de Estilo en Cascada* (en inglés, *Cascading Style Sheets*) (CSS) (Lie and Bos, 1996) es un lenguaje de hojas de estilo usado para describir la presentación de un documento HTML permitiendo la separación del contenido del documento (escrito en HTML) de la presentación del documento (escrito en CSS). Esta separación permite mejorar la

accesibilidad del contenido, proveyendo más flexibilidad y control en la especificación de las características presentacionales, reduciendo así la complejidad y la repetición en el contenido estructural.

Además de utilizarse para la publicación de contenido en la forma de documentos web, HTML puede usarse para construir *front-ends* para aplicaciones basadas en la web. En tales sistemas, la entrada del usuario es realizada a través de *formularios HTML*. Los formularios HTML se utilizan así para seleccionar diferentes tipos de entrada de usuario. Básicamente, un formulario es un área que puede contener *elementos de formulario*. Entre los elementos de formulario podemos encontrar campos de texto (*text field*), campos de área de texto (*textarea fields*), menús descolgables, radio botones, cajas de verificación (*checkbox*), etc.

Para extender las capacidades de los formularios web simples, se pueden utilizar *guiones* JavaScript para agregar validación e interactividad sin incrementar la sobrecarga en el lado del servidor.

2.2.2. Extensible Markup Language

El *Lenguaje de Marcado Extensible* (en inglés, *Extensible Markup Language* o XML) (McGrath, 1998) es un lenguaje de marcado general para crear lenguajes de marcado de propósito específico, capaces de describir muchos tipos diferentes de datos. En particular, XML es un subconjunto simplificado del *Lenguaje Estándar de Marcado Generalizado* SGML (en inglés, *Standard Generalized Markup Language*o SGML). Su propósito primario es facilitar el compartir datos a través de diferentes sistemas, en particular sistemas conectados a través de Internet. Dichos sistemas deben acordar el formato común de los documentos XML que comparten.

El propósito de una *Definición de Tipo de Documento* (en inglés, *Document Type Definition* o DTD) es la definición de los bloques de construcción legales de un documento XML. Define la estructura del documento con una lista de elementos legales. Los DTS no se definen como documentos XML sino con un lenguaje diferente. Luego, *XML Schema* fue desarrollado como una alternativa a los DTDs basada en XML.

El *Lenguaje de Marcado de Hipertexto Extensible* (en inglés, *Extensible HyperText Markup Language* o XHTML) (Pemberton et al., 2002) es una versión más estricta y más limpia de HTML pensada para reemplazar a HTML. Básicamente, XHTML es HTML redefinido como una aplicación XML. En el marco de XHTML, los formularios web también han sido redefinidos como una aplicación XML: el lenguaje *XForms* usa XML para la definición de datos y HTML o XHTML para la presentación de datos. El lenguaje XForms separa la lógica de los datos de un formulario de su presentación. De esta manera, los datos de XForms pueden definirse independientemente de cómo el usuario final va a interactuar con la aplicación (*e.g.*, en una computadora de escritorio, un PDA o un

teléfono celular).

2.2.3. Resource Description Framework

El *Marco de Descripción de Recursos* (en inglés, *Resource Description Framework* o RDF) (Manola and Miller, 2004) es un estándar de la W3C para describir recursos en la Web, tales como el título, autor, fecha de modificación, contenido e información de derechos de autor de una página Web. RDF fue diseñado para proveer una manera común de describir información de tal manera que pueda ser leída y entendida por aplicaciones de computadora. Las descripciones RDF no han sido diseñadas para ser mostradas en la Web. Los documentos RDF se escriben en un lenguaje XML llamado RDF/XML. Al utilizar XML, la información RDF puede ser fácilmente intercambiada entre diferentes tipos de computadoras que usen distintos tipos de sistemas operativos y lenguajes de aplicación. Mientras que XML provee soporte sintáctico para RDF, la teoría de grafos provee soporte semántico para RDF.

El elemento base del *modelo RDF* es la *tripla*: un recurso (el *sujeto*) es enlazado a otro recurso (el *objeto*) por medio de un arco etiquetado con un tercer recurso (el *predicado*) (Champin, 2001). Diremos que "sujeto" tiene una propiedad "predicado" valuada por "objeto". Por ejemplo, la tripla ⟨*http://cs.uns.edu.ar/~sag/index.htm, http://purl.org/DC/Creator, mailto:sag@cs.uns.edu.ar*⟩ puede ser interpretada como "Sergio Gómez es el creador de la página index.htm". El conjunto de todas las triplas resulta en un grafo dirigido, cuyos nodos y arcos están etiquetados con URIs calificadas. Este grafo describe recursos con clases, propiedades y valores.

Además, RDF también requiere de una manera de definir clases y propiedades específicas de las aplicaciones. Dichas clases y propiedades específicas de las aplicaciones deben definirse usando extensiones como *RDF Schema* (RDFS). En realidad, RDFS no provee clases y propiedades específicas de aplicaciones, sino que RDFS provee el marco para describir dichas clases y propiedades. Las clases en RDFS son muy similares a las clases en lenguajes de programación orientados a objetos; en particular, RDFS permite que los recursos sean definidos como instancias de clases, y subclases de clases.

2.2.4. DARPA Agent Markup Language

RDF y RDFS proveen el conjunto de características básicas para el modelado de la información. Esta simplicidad hace que sea una especie de lenguaje ensamblador de la web sobre el cual casi cualquier otro método de modelado de información puede apoyarse. Sin embargo, en respuesta a la necesidad de tipos de datos, una expresión consistente para definir tipos enumerados así como otras facilidades, el *Lenguaje de Marcado DARPA* (en inglés, *DARPA Agent Markup Language* o DAML) fue lanzado como *DAML-ONT* (McGuiness et al., 2003), un lenguaje simple para expresar definiciones de clase más sofisti-

cadas que las permitidas por RDFS. El grupo DAML unió esfuerzos con el grupo de la *Capa de Inferencia de Ontologías* (en inglés, *Ontology Inference Layer* u OIL) (Decker et al., 2000), un proyecto con el objetivo de proveer un mecanismo de clases más sofisticado, usando construcciones de la IA basadas en la teoría de marcos (frames). El resultado de dichos esfuerzos es *DAML+OIL* (Connolly et al., 2001) que también agrega facilidades para tipos de datos basado en definiciones de tipos provistas por medio del *Lenguaje de Definición de Esquemas XML* (en inglés, *XML Schema Definition Language* o XSDL).

El modelo de DAML+OIL está basado en *Lógicas para la Descripción* muy expresivas (en inglés, *Description Logics*). Las Lógicas para la Descripción (DL) (Baader et al., 2003) son una familia conocida de formalismos de representación de conocimiento. Están basadas en las nociones de *conceptos* (predicados unarios, clases) y *roles* (relaciones binarias), y principalmente se caracterizan por constructores que permiten construir conceptos y roles complejos a partir de otros más simples. El poder expresivo de un sistema de DL está determinado por las construcciones disponibles para formar descripciones de conceptos, y por la manera en que estas descripciones pueden utilizarse en los componentes terminológicos (Tbox) y asercionales (Abox) del sistema (para más detalles, referirse a la Sección 2.3).

2.2.5. Ontology Web Language

El *Lenguaje de Ontologías Web* (OWL, en inglés, *Web Ontology Language*) (McGuiness and van Harmelen, 2004; Smith et al., 2004; Bechhofer et al., 2003, 2004) está pensado para ser usado cuando la información contenida en documentos necesita ser procesada por aplicaciones, en contraposición a situaciones donde el contenido sólo necesita ser presentado a humanos. OWL puede ser utilizado explícitamente para representar el significado de términos en vocabularios y las relaciones entre dichos términos. Esta representación de términos y sus relaciones es llamada una ontología. OWL tiene más facilidades para expresar significado y semántica que XML, RDF, y RDF-Schema, y de esta manera OWL va más allá de dichos lenguajes en su habilidad de representar contenido Web interpretable por máquinas.

OWL tiene una relación muy clara en el escenario de los lenguajes de representación de conocimiento en la Web. XML provee una sintaxis para estructurar documentos, pero no impone restricciones semánticas en el significado de dichos documentos. XML Schema es un lenguaje para restringir la estructura de los documentos XML y, también, para extender XML con tipos de datos. RDF es modelo de datos para describir objetos (o *recursos*) y relaciones entre ellos; provee un semántica simple para este modelo de datos, y, dicho modelo de datos puede ser representado en una sintaxis XML. RDF Schema es un vocabulario para describir propiedades y clases de recursos RDF con una semántica para jerarquías de generalización sobre tales propiedades y clases. OWL agrega más vocabulario

para describir propiedades y clases: entre otras, relaciones entre clases (*e.g.*, disyunción), cardinalidad (*e.g.*, exactamente uno), igualdad, tipado rico de propiedades, características de propiedades (*e.g.*, simetría), y clases enumeradas.

El lenguaje OWL contiene tres sub-lenguajes de expresividad creciente: *OWL Lite*, *OWL DL* y *OWL Full*. Como en el caso de RDF, al usar XML, la información expresada en OWL puede ser fácilmente intercambiada entre diferentes tipos de computadoras usando distintos tipos de sistemas operativos y lenguajes de aplicación.

Como OWL hace la llamada *suposición de mundo abierto*, se asume que las descripciones de los recursos no se consideran confinadas a un archivo simple o ámbito. Por lo tanto, mientras que la clase C_1 puede ser definida originalmente en la ontología Σ_1, puede ser extendida en otras ontologías. Las consecuencias de proposiciones adicionales a C_1 son monotónicas. La nueva información no puede retractar información previa.

Es importante notar que la Recomendación de la W3C especifica que es responsabilidad del ingeniero de conocimiento determinar si el agregado de nueva información en la ontología produce o no inconsistencias. Citando textualmente a (McGuiness and van Harmelen, 2004):

> *La nueva información puede ser contradictoria, pero hechos y derivaciones pueden ser sólo agregados, nunca eliminados. La posibilidad de tales contradicciones es algo que el diseñador de una ontología necesita tomar en consideración. La Recomendación de la W3C espera que el soporte brindado por herramientas ayude a detectar tales casos.*

Como mencionamos anteriormente, las ontologías OWL pueden ser categorizadas en tres sublenguajes: OWL-Lite, OWL-DL y OWL-Full. Una característica definitoria de cada lenguaje es su expresividad. OWL-Lite es el lenguaje menos expresivo mientras que OWL-Full es el más expresivo y, la expresividad de OWL-DL se halla entre la de éstos últimos. Así, OWL-DL puede ser considerado como una extensión de OWL-Lite y OWL-Full como una extensión de OWL-DL. A continuación, damos un breve resumen de cada uno de ellos:

- **OWL-Lite:** *OWL-Lite* es el sublenguaje sintácticamente más simple, da soporte a aquellos usuarios que necesitan una jerarquía de clasificación y restricciones simples. Por ejemplo, mientras soporta restricciones de cardinalidad, sólo permite valores de cardinalidad de 0 o 1. OWL básicamente ha sido diseñado para proveer un camino rápido para la migración de *thesauri* y otras taxonomías.

- **OWL-DL:** *OWL DL* da soporte a aquellos usuarios que quieren el máximo de expresividad pero reteniendo completitud computacional (todas las conclusiones están garantizadas como computables) y decidabilidad (todas las computaciones terminan

en un tiempo finito). OWL DL incluye todas las construcciones del lenguaje OWL, pero pueden ser usadas bajo ciertas restricciones (*e.g.*, mientras una clase puede ser subclase de muchas clases, una clase no puede ser una instancia de otra clase). El nombre OWL DL proviene de su correspondencia con la Lógicas para la Descripción (en inglés, *Description Logics* o DL), las cuales son un campo de investigación que ha estudiado las lógicas que forman la fundación formal de OWL. Como veremos en la Sección 2.3.3, las DL son un fragmento decidible de la Lógica de Primer Orden (FOL) y, por lo tanto, capaces de someterse a técnicas de razonamiento automatizado. Por lo tanto, es posible computar automáticamente la jerarquía de clases y chequear inconsistencias en una ontología que conforme a OWL-DL.

- **OWL-Full:** *OWL-Full* es el sublenguaje más expresivo. Por ejemplo, OWL permite representar a las clases como individuos, dando lugar a la posibilidad de expresar enunciados paradójicos. Está pensado para ser usado en situaciones donde una alta expresividad es más importante que ser capaz de garantizar la decidibilidad o completitud computacionales del lenguaje. Por lo tanto, no es posible realizar razonamiento automatizado en una ontología OWL-Full. OWL Full está pensado para usuarios que quieren máxima expresividad y la libertad sintáctica de RDF pero sin garantías computacionales. Por ejemplo, en OWL Full una clase puede ser tratada simultáneamente como una colección de individuos y como un individuo. También OWL Full permite aumentar el significado del vocabulario (RDF u OWL) predefinido. De acuerdo a sus creadores, es poco probable que algún software de razonamiento vaya a proveer soporte completo de razonamiento para cada característica de OWL.

La elección entre OWL-Lite y OWL-DL puede estar basada en el hecho de si las construcciones de OWL-Lite son suficientes o no. La elección entre OWL-DL y OWL-Full puede estar basada en si es o no importante representar clases de clases (metaclases) o en si hay que asegurar la decidibilidad de los resultados obtenibles a partir de la ontología. A continuación extendemos el resumen de las características principales de cada uno de ellos presentado más arriba; nuestra presentación está basada en los trabajos (McGuiness and van Harmelen, 2004; Smith et al., 2004; Bechhofer et al., 2004).

OWL Lite

OWL Lite provee constructores de clase, como *Class*, que permiten definir clases. una clase define un grupo de individuos que pertenecen juntos porque comparten ciertas propiedades. *Thing* es la clase más general y *Nothing* es la clase vacía. Las jerarquías de clases pueden ser creadas haciendo uno o más afirmaciones que una clase es subclase de otra clase (con el constructor *rdfs:subClassOf*).

Las propiedades pueden ser usadas para establecer propiedades entre individuos o de individuos a valores de datos, por ejemplo enteros (constructor *rdf:Property*). Jerarquías

de propiedades pueden ser creadas haciendo una o más afirmaciones que una propiedad es subpropiedad de una o más propiedades (constructor *rdfs:subPropertyOf*). El dominio de una propiedad limita los individuos a los cuales la propiedad puede ser aplicada (*rdfs:domain*). Si una propiedad relaciona un individuo con otro individuo, y la propiedad tiene una clase como uno de sus dominios, entonces el individuo debe pertenecer a dicha clase. El rango de una propiedad (*rdfs:range*) limita los individuos que la propiedad puede tomar como su valor. Si la propiedad relaciona un individuo con otro individuo, y la propiedad tiene un clase como su rango, entonces el otro individuo debe pertenecer a la clase rango. Los individuos son instancias de clases (*Individual*), y las propiedades pueden ser usadas para relacionar un individuo con otro.

Además, OWL-Lite permite definir nuevas clases relacionándolas con clases creadas previamente por medio del uso de los llamados constructores de igualdad. Por ejemplo, dos clases pueden establecerse como equivalentes por medio del constructor *equivalentClass*; dos clases son equivalentes si tienen las mismas instancias. Dos propiedades pueden establecerse como equivalentes (constructor *equivalentProperty*); dos propiedades son equivalentes si relacionan a un individuo con el mismo conjunto de individuos. Dos individuos pueden indicarse como el mismo individuo (constructor *sameAs*). Un individuo se puede indicar como diferente de otros individuos con el constructor *differentFrom*. Finalmente, un número de individuos pueden ser indicados como mutuamente diferentes (constructor *AllDifferent*).

También, OWL-Lite permite especificar características de las propiedades. Una propiedad puede establecerse como la inversa de otra propiedad con el constructor *inverseOf*. Las propiedades pueden establecerse como transitivas (constructor *TransitiveProperty*), simétricas (constructor *SymmetricProperty*), tener un único valor (constructor *FunctionalProperty*) o ser funcionales inversas (constructor *InverseFunctionalProperty*); si una propiedad es funcional inversa entonces la inversa de la propiedad es funcional.

OWL Lite permite restricciones sobre cómo las propiedades pueden ser usadas por las instancias de una clase. La restricción *allValuesFrom* es establecida sobre una propiedad con respecto a una clase. Esto significa que esta clase particular tiene una restricción de rango local asociada con ella. Así, si una instancia de la clase es relacionada con la propiedad a un segundo individuo, entonces el segundo individuo puede ser inferido como perteneciente a una instancia de la clase del rango local. Las restricción *someValuesFrom* es establecida sobre una propiedad con respecto a una clase. Una clase particular puede tener una restricción sobre una propiedad donde al menos un valor de la propiedad debe ser de un cierto tipo.

OWL Lite incluye una forma limitada de restricciones de cardinalidad. Las restricciones de cardinalidad son limitadas porque sólo permiten sentencias concernientes a valores de cardinalidad iguales a 0 o 1. Con la restricción *minCardinality*, la cardinalidad es establecida sobre una propiedad con respecto a una clase particular . Si *minCardinality* de

1 es establecida sobre una propiedad con respecto a una clase, entonces cualquier instancia de la clase estará relacionada con *al menos* un individuo para dicha propiedad. Esta restricción es otra manera de decir que se requiere que la propiedad tenga un valor para todas las instancias de la clase. Con la restricción *maxCardinality*, la cardinalidad es establecida sobre una propiedad con respecto a una clase particular. Si *maxCardinality* de 1 es establecida sobre una propiedad con respecto a una clase, entonces cualquier instancia de la clase estará relacionada con *a lo sumo* un individuo para dicha propiedad. La restricción *cardinality* es usada como una conveniencia para establecer que una propiedad sobre una clase tiene ambas *minCardinality* y *maxCardinality* igual a 0 o a 1.

OWL Lite contiene un constructor de intersección pero limita su uso. El constructor *intersectionOf* es utilizado en OWL Lite para definir intersecciones de clases nombradas y restricciones.

Con respecto al soporte de tipos de datos primitivos, OWL usa los mecanismos de RDF para valores de datos. Si bien existen varios tipos de datos incluyendo diversas variedades de representación de números, booleanos, fechas y cadenas, nos remitiremos solamente al hecho de que OWL soporta al menos los tipos entero `xsd:integer` y cadena `xsd:string`.

Constructores de OWL DL y OWL Full

OWL DL y OWL Full proveen algunos constructores adicionales a los provistos por OWL Lite. Mediante el constructor *oneOf*, las clases pueden ser descriptas como la enumeración de los individuos que pertenecen a la clase; los miembros de la clase son exactamente los elementos especificados en la enumeración. Por medio del constructor *hasValue*, se puede requerir que una propiedad tenga exactamente a un cierto individuo como su valor. Las clases se pueden establecer como disjuntas una de otra con el constructor *disjointWith*. OWL DL y OWL Full permiten combinaciones booleanas arbitrarias de clases y restricciones; los constructores *unionOf*, *complementOf* y *intersectionOf* permiten especificar nuevas clases como la unión de dos clases existentes, el complemento de una clase existente o la intersección de dos clases existentes, respectivamente. Mientras que en OWL Lite, las cardinalidades están restringidas a 0 y 1, OWL DL y Full permiten que las cardinalidades sean un número natural (con una semántica diferente de los constructores *minCardinality*, *maxCardinality* y *cardinality* provistos por OWL Lite).

2.3. Lógicas para la Descripción

Las *Lógicas para la Descripción* (DL) son una familia de formalismos de representación de conocimiento bien conocidos (Baader et al., 2003), que proveen una familia de lenguajes de representación de conocimiento y mecanismos de inferencia. El nombre *Lógicas para la Descripción* está motivado por el hecho que las nociones importantes del dominio son

descriptas por *descripciones de conceptos*, es decir, expresiones que son construidas a partir de conceptos atómicos (predicados unarios) y roles atómicos (predicados binarios) usando los conceptos y constructores de roles provistos por la DL particular. El estudio de estas lógicas es relevante en el marco de este trabajo de Tesis pues las mismas subyacen la semántica de los lenguajes de representación de conocimiento de la Web Semántica, que permiten modelar la información del mundo real. Sin embargo, también son importantes debido a que las bases de datos relacionales, orientadas a objetos y diagramas UML se pueden reducir a las mismas (Calvanese et al., 1998).

Las DL están basadas en las nociones de *conceptos* (predicados unarios, clases) y *roles* (relaciones binarias), y están principalmente caracterizadas por constructores que permiten construir conceptos complejos y roles a partir de otros atómicos. El poder expresivo de un sistema DL está determinado por los constructores disponibles para construir descripciones de conceptos, y por la manera en que estas descripciones pueden ser usadas en las componentes *terminológicas* (Tbox) y *asercionales* (Abox) del sistema.

Los mecanismos de inferencia soportan tareas comunes de razonamiento del mundo real: la clasificación de conceptos e individuos. Los sistemas de representación de conocimiento basados en DL brindan a sus usuarios varias capacidades de inferencia que deducen conocimiento implícito a partir del conocimiento representado explícitamente. Por ejemplo, el algoritmo de *subsunción* permite determinar relaciones de subconcepto-superconcepto: un concepto C está subsumido por otro concepto D si y sólo si todas las instancias de C son también instancias de D; es decir, la primera descripción es siempre interpretada como un subconjunto de la segunda descripción. También, es importante resolver el problema de la *pertenencia*: un individuo a pertenece a un concepto C si $C(a)$ es satisfacible en todas las interpretaciones del concepto C.

Es de notar que el avance importante provisto por las DL respecto de formalismos anteriores (como las redes de herencia estructuradas (Brachman, 1979)) está dado porque el sistema es capaz de inferir las relaciones de subclase-superclase implícitas en la base de conocimiento. En las redes de herencia, el usuario debe establecer explícitamente los enlaces IS-A entre conceptos atómicos. En cambio, en las DL, el usuario solamente expresa relaciones entre conceptos (que no son necesariamente conceptos atómicos) y el sistema es capaz de inferir la jerarquía de herencia implícita.

Observación 2.3.1 *Los razonadores sobre Lógicas para la Descripción como* RACER *soportan la Hipótesis de Mundo Abierto (Haarslev and Möller, 2004, pág. 19). Es decir, que aquello que no se puede probar como verdadero no es asumido como falso.*

2.3.1. Lenguaje de representación de conocimiento

Las Lógicas para la Descripción se distinguen por los constructores para formar expresiones de clases que brindan. El lenguaje básico es llamado $\mathcal{AL}$ (por *attributive language*

o *lenguaje atributivo*). Los otros lenguajes de la familia son extensiones de este lenguaje básico y son nombrados con letras especiales indicando los constructores extra que los conforman.

A continuación presentamos el lenguaje que utilizaremos, el cual es el llamado $\mathcal{ALCEU}$, donde $\mathcal{C}$ es por complemento de concepto, $\mathcal{E}$ por restricción existencial ilimitada y $\mathcal{U}$ por unión de conceptos. Sean C y D conceptos y R un nombre de rol. Las descripciones de conceptos están construidas a partir de nombres de conceptos usando los constructores de conjunción ($C \sqcap D$), disyunción ($C \sqcup D$), negación ($\neg C$), restricción existencial ($\exists R.C$), y restricción de valor ($\forall R.C$).

Ejemplo 2.3.1 *La siguiente expresión denota el conjunto de los individuos que no son capaces de volar "$\neg$flies"; la siguiente expresión DL denota la intersección de los autos de carrera y los autos de calle. "raceCar $\sqcap$ streetCar". Ahora presentamos una expresión de concepto que denota al conjunto de individuos que son autos de carrera o bicicletas fijas "raceCar $\sqcup$ stationaryBike". La siguiente expresión denota al conjunto de individuos que fueron operados por un cirujano genético: "$\exists$operatedBy.geneticSurgeon". Ahora presentamos una expresión que denota a todos los objetos cuyas ruedas son de goma: "$\forall$esRuedaDe.EsDeGoma". Nótese que "operatedBy" y "esRuedaDe" son relaciones binarias mientras que "flies", "raceCar", "geneticSurgeon" y "EsDeGoma" son conceptos.*

Para definir la semántica de las descripciones de conceptos, los conceptos se interpretan como subconjuntos de un dominio de interés, y los roles como relaciones binarias sobre este dominio. Una interpretación $\mathcal{I}$ consiste de un conjunto no vacío $\Delta^{\mathcal{I}}$ (el dominio de $\mathcal{I}$) y una función $\cdot^{\mathcal{I}}$ (la función de interpretación de $\mathcal{I}$) que asigna a cada nombre de concepto A un subconjunto $A^{\mathcal{I}}$ de $\Delta^{\mathcal{I}}$, y a cada nombre de rol R un subconjunto $R^{\mathcal{I}}$ de $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. La función de interpretación es extendida a descripciones arbitrarias de conceptos como se muestra a continuación:

- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$;
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$;
- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$;
- $(\exists R.C)^{\mathcal{I}} = \{x | \exists y \text{ tal que } (x, y) \in R^{\mathcal{I}} \text{ e } y \in C^{\mathcal{I}}\}$, y
- $(\forall R.C)^{\mathcal{I}} = \{x | \forall y, (x, y) \in R^{\mathcal{I}} \text{ implica } y \in C^{\mathcal{I}}\}$.

Además, las expresiones $\top$ y $\bot$ son abreviaturas para $C \sqcup \neg C$ y $C \sqcap \neg C$, respectivamente.

Nótese que la semántica respeta las equivalencias $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$, $C \sqcap D \equiv \neg(\neg C \sqcup \neg D)$ (llamadas reglas de De Morgan), $\exists R.C \equiv \neg\forall R.\neg C$ y $\forall R.C \equiv \neg\exists R.\neg C$. Por lo tanto, la unión y la cuantificación existencial completa nos da la habilidad de expresar

la negación de conceptos a través de su equivalente forma normal de negación (en inglés, *negation normal form*). Para obtener la forma normal de negación, se deben *empujar* los símbolos de negación hacia adentro en las descripciones, usando las reglas de De Morgan y las reglas para los cuantificadores; así, cuando una expresión se encuentra en forma normal negada, la negación ocurre solamente en el frente de nombres de conceptos.

Definición 2.3.1 (Forma normal de negación (Krötzch et al., 2006)) *Para una descripción de conceptos C dada, la forma normal de la negación* $\mathsf{NNF}(C)$ *se define recursivamente de la siguiente manera:* [3]

$$
\begin{array}{lll}
\mathsf{NNF}(C) & =_{def} & C \text{ para todo } C \in \{\top, \bot, A, \neg A\} \\
\mathsf{NNF}(\neg\top) & =_{def} & \bot \\
\mathsf{NNF}(\neg\bot) & =_{def} & \top \\
\mathsf{NNF}(\neg\neg C) & =_{def} & \mathsf{NNF}(C) \\
\mathsf{NNF}(C \sqcap D) & =_{def} & \mathsf{NNF}(C) \sqcap \mathsf{NNF}(D) \\
\mathsf{NNF}(\neg(C \sqcap D)) & =_{def} & \mathsf{NNF}(\neg C) \sqcup \mathsf{NNF}(\neg D) \\
\mathsf{NNF}(C \sqcup D) & =_{def} & \mathsf{NNF}(C) \sqcup \mathsf{NNF}(D) \\
\mathsf{NNF}(\neg(C \sqcup D)) & =_{def} & \mathsf{NNF}(\neg C) \sqcap \mathsf{NNF}(\neg D) \\
\mathsf{NNF}(\forall R.C) & =_{def} & \forall R.\mathsf{NNF}(C) \\
\mathsf{NNF}(\neg\forall R.C) & =_{def} & \exists R.\mathsf{NNF}(\neg C) \\
\mathsf{NNF}(\exists R.C) & =_{def} & \exists R.\mathsf{NNF}(C) \\
\mathsf{NNF}(\neg\exists R.C) & =_{def} & \forall R.\mathsf{NNF}(\neg C) \\
\mathsf{NNF}(\leq nR.C) & =_{def} & \leq nR.\mathsf{NNF}(C) \\
\mathsf{NNF}(\neg \leq nR.C) & =_{def} & \geq (n+1)R.\mathsf{NNF}(C) \\
\mathsf{NNF}(\geq nR.C) & =_{def} & \geq nR.\mathsf{NNF}(C) \\
\mathsf{NNF}(\neg \geq nR.C) & =_{def} & \leq (n-1)R.\mathsf{NNF}(C)
\end{array}
$$

Ejemplo 2.3.2 *Sean R un nombre de relación y C, D nombres de conceptos atómicos. Consideremos la expresión denotando un concepto complejo $\neg\exists R.(\neg(C \sqcap \neg D))$. Dicha expresión es normalizada como: $\forall R.(C \sqcap \neg D)$. En cambio, la expresión $\neg\exists R.(C \sqcup \neg D)$ es normalizada como: $\forall R.(\neg C \sqcap D)$.*

Ejemplo 2.3.3 (Adaptado de (Baader et al., 2003)) *Sea la siguiente descripción de concepto:*

$$(\exists R.A) \sqcap (\exists R.B) \sqcap \neg(\exists R.(A \sqcap B)).$$

Su forma normal negada está dada por la siguiente descripción de concepto:

$$(\exists R.A) \sqcap (\exists R.B) \sqcap \forall R.(\neg A \sqcup \neg B).$$

[3] Las últimas cuatro ecuaciones corresponden a la extensión $\mathcal{N}$ que se presenta más abajo.

La extensión $\mathcal{N}$ agrega las restricciones numéricas, notadas como $\geq nR$ (restricción al menos) y como $\leq nR$ (restricción a lo sumo), donde n es un entero no negativo. Éstas son interpretadas como:[4]

$$\begin{array}{rcl}(\geq nR)^{\mathcal{I}} & = & \left\{\ a \in \Delta^{\mathcal{I}} | \#\{b | (a,b) \in R^{\mathcal{I}}\} \geq n\ \right\}, \text{ y} \\ (\leq nR)^{\mathcal{I}} & = & \left\{\ a \in \Delta^{\mathcal{I}} | \#\{b | (a,b) \in R^{\mathcal{I}}\} \leq n\ \right\}.\end{array}$$

Ejemplo 2.3.4 *La siguiente expresión de concepto denota a todos los individuos que tienen al menos cuatro ruedas: "($\geq$ 4 esRuedaDe)", mientras que la siguiente denota al conjunto de individuos que tienen a lo sumo 18 ruedas "($\leq$ 18 esRuedaDe)".*

Otras extensiones para las DL básicas incluyen roles inversos y transitivos, notados como P^- and P^+, respectivamente (Horrocks and Sattler, 1999). Un par $(x,y) \in (P^-)^{\mathcal{I}}$ si y sólo si $(y,x) \in P^{\mathcal{I}}$. Un par $(x,z) \in (P^+)^{\mathcal{I}}$ si y sólo si existe y tal que $(x,y) \in P^{\mathcal{I}}$ e $(y,z) \in P^{\mathcal{I}}$.

Ejemplo 2.3.5 *Por ejemplo, podemos tratar en forma anónima el rol "es hijo de" representándolo como la inversa del rol "es padre de" con la notación: "esPadreDe$^-$". Por otro lado, si consideramos los árboles y sus nodos, podemos representar la transitividad del rol "esDencendienteDe" que relaciona dos nodos con "esDescendienteDe$^+$".* [5]

Otras extensiones permiten *nombres de individuos* (llamados también *nominales*) en el lenguaje de descripción. El más básico es el constructor de conjuntos (o "*one-of*"), escrito como:

$$\{a_1, \ldots, a_n\},$$

donde $a_1, \ldots, a_n$ son nombres de individuos. Tal concepto es interpretado como:

$$\{a_1, \ldots, a_n\}^{\mathcal{I}} = \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}.$$

Ejemplo 2.3.6 *Con el constructor one-of se puede definir el concepto de los colores primarios como $\{amarillo, rojo, azul\}$.*

Una *ontología* consiste de dos conjuntos finitos y mutuamente disjuntos. Un conjunto *Tbox* que introduce la *terminología* y un conjunto *Abox* o *caja asercional* que contiene hechos acerca de objetos particulares en el dominio de aplicación. Las sentencias de la Tbox son de la forma $C \sqsubseteq D$ (*inclusiones*) y $C \equiv D$ (*igualdades*), donde C y D son descripciones de conceptos (posiblemente complejas). La semántica de las sentencias de la Tbox es como sigue. Una interpretación I satisface $C \sqsubseteq D$ si y sólo si $C^I \subseteq D^I$, I satisface $C \equiv D$ si y sólo si $C^I = D^I$.

[4] El operador $\#\cdot$ denota la cardinalidad de conjuntos.

[5] En realidad, existe una notación fija para realizar esto, que consiste en usar un axioma de inclusión de clase de la forma "*esDescendienteDe* $\sqsubseteq$ *esDescendienteDe*$^+$".

Los objetos en la Abox son referidos por un número finito de *nombres de individuos* y estos nombres pueden ser usados en dos tipos de sentencias asercionales: *aserciones de concepto* del tipo $a : C$ y *aserciones de rol* del tipo $\langle a, b\rangle : R$, donde C es una descripción de concepto, R es un nombre de rol, y a y b son nombres de individuos. Una interpretación I *satisface* la aserción $a : C$ si y sólo si $a^I \in C^I$, y *satisface* $\langle a, b\rangle : R$ si y sólo si $(a^I, b^I) \in R^I$. Una interpretación I es un *modelo* de una sentencia ϕ DL (en la Tbox o en la Abox) si y sólo si satisface la sentencia, y es un modelo de una ontología DL Σ si satisface cada sentencia en Σ. Una sentencia DL ϕ se *sigue* de una ontología Σ, escrito como $\Sigma \models \phi$, si y sólo si cada modelo de Σ es también un modelo de ϕ.

Observación 2.3.2 *Es de notar que en algunos textos,* e.g.*(Baader et al., 2003), las sentencias de aserción son notadas como "$C(a)$" (aserción de individuos) y "$R(a, b)$" (aserción sobre relaciones) y en otros como "$a : C$" y "$\langle a, b\rangle : R$", respectivamente. En esta presentación, usaremos mayormente la segunda notación porque permite expresar aserciones sobre conceptos y relaciones no necesariamente atómicos. Además, la usamos para diferenciarla de los hechos en Lógica de Primer Orden y, en particular, de la Programación en Lógica Rebatible. Sin embargo, cuando no surja confusión las usaremos en forma indistinta.*

A continuación, presentaremos una ontología con la cual mostraremos los distintos constructores de las Lógicas para la Descripción presentados previamente. Luego, en las secciones siguientes, utilizaremos esta ontología para ejemplificar las distintas tareas de razonamiento que es posible llevar a cabo. Nótese que esta ontología fue codificada en el lenguaje RACER y las tareas de razonamiento fueron corroboradas con el motor de inferencia provisto por tal lenguaje de programación (Haarslev and Möller, 2001, 2004).

Ejemplo 2.3.7 *En las Figuras 2.2 y 2.3, presentamos una ontología $\Sigma_{2.3.7} = (T_{star\text{-}wars}, A_{star\text{-}wars})$ sobre parte del universo contenido en la saga de la Guerra de las Galaxias.* [6] *Parte de esta ontología es una adaptación de la ontología de (Baader et al., 2003, Fig. 2.2, Pág. 52) sobre relaciones familiares.*

A continuación discutimos el significado de los axiomas de la Tbox $T_{star\text{-}wars}$, que es presentada en la Figura 2.2. Los axiomas r_1 al r_{10} describen relaciones familiares. En particular: el axioma r_1 expresa que las mujeres son personas femeninas; el axioma r_2 dice que los hombres son personas masculinas; r_3 indica que el conjunto de los hombres y las mujeres son disjuntos; r_4 establece que las madres son aquellas mujeres que tienen al menos un hijo mientras que r_5 dice que los padres son aquellos hombres que tienen un hijo; r_6 expresa que los hijos son individuos masculinos que tienen un progenitor mientras que r_7 dice que las hijas son las féminas que tienen un progenitor; r_8 dice que los progenitores es el conjunto de los padres junto con el de las madres; el axioma r_9 establece que tener un hijo y ser progenitor son relaciones inversas.

[6] Véase la página web http://www.starwars.com.

Por otro lado, r_{10} dice que la unión de las ciudades, planetas y naves está contenida en el conjunto los lugares; r_{11} expresa que Alderaan, Bespin, Corellia y Tattoine son planetas; r_{12} establece que la Estrella de la Muerte y el Halcón Milenario son naves; r_{13} dice que Cloud City es una ciudad; r_{14} indica que un lugar para aterrizar no debe estar destruido; [7] *r_{15} dice que el conjunto de las cosas destruidas contiene a aquellas que fueron destruidas por un arma; r_{16} expresa que los roles "ciudad origen de" y "pasajero de" están contenidos en el rol "conoce un lugar"; r_{17} dice que las ciudades están ubicadas en un planeta; r_{18} establece que "comandante de" y "comandado por" son roles inversos; r_{19} expresa que un nave es comandada por alguien, y r_{20} dice que un comandante comanda a algo.*

La Abox $A_{star\text{-}wars}$ es presentada en la Figura 2.3. Las sentencias a_1 y a_2 establecen que Leia Organa es una persona de sexo femenino; a_3–a_{10} dicen que Luke Skywalker, Lando Calrissian, Han Solo y Anakin Skywalker son personas de sexo masculino; a_{11}–a_{14} dicen que los planetas natales de Han Solo, Luke Skywalker, Leia Organa y Anakin Skywalker son Corellia, Tattoine, Alderaan y Tattoine, respectivamente. Las aserciones a_{15} y a_{16} dicen que Anakin Skywalker es el padre de Luke Skywalker y Leia Organa, mientras que a_{17} y a_{18} expresan que Luke Skywalker y Leia Organa son pasajeros del Halcón Milenario. La aserción a_{19} expresa que Cloud City está ubicada en el planeta Bespin; a_{20} y a_{21}, que la Estrella de la Muerte es un arma y una nave; a_{22}, que el Halcón Milenario es una nave; a_{23} y a_{24}, que Han Solo y Lando Calrissian son comandantes del Halcón Milenario; finalmente, a_{25} dice que el planeta Alderaan fue destruido por la Estrella de la Muerte.

Ejemplo 2.3.8 *Considere la siguiente ontología $\Sigma_{2.3.8} = (T, A)$ sobre el dominio de los vehículos:*

$$T = \left\{ \begin{array}{l} Air\_Vehicle \sqcup Land\_Vehicle \sqcup Water\_Vehicle \sqsubseteq Vehicle \\ Car \sqcup Tank \sqcup Truck \sqsubseteq Land\_Vehicle \\ Baloon \sqcup Helicopter \sqcup Plane \sqsubseteq Air\_Vehicle \\ Boat \sqsubseteq Water\_Vehicle \\ Ferry \sqcup Kayak \sqcup Yacht \sqsubseteq Boat \end{array} \right\}$$

$$A = \left\{ \begin{array}{l} scania : Truck \\ corsa : Car \\ boeing747 : Plane \end{array} \right\}$$

La Tbox T posee axiomas con el siguiente significado: los vehículos acuáticos, aéreos y terrestres son vehículos; los autos, camiones y tanques son vehículos terrestres; los aviones, globos y helicópteros son vehículos aéreos; los botes son vehículos acuáticos, y, los ferries, kayaks y veleros son tipos de botes. La Abox A posee aserciones estableciendo que: Scania es un camión, Corsa es un auto y Boeing 747 es un avión.

Ejemplo 2.3.9 *Considere la siguiente terminología $T_{2.3.9}$, donde:*

[7] La notación $C \sqsupseteq D$ es un abuso de notación realizado con respecto a $D \sqsubseteq C$. Esta notación también es utilizada por otros autores del área de la Web Semántica (*e.g.*, (Cuenca Grau et al., 2007, Ejemplo 4)).

$$T_{2.3.9} = \left\{ \begin{array}{l} bird \sqsubseteq flies \\ penguin \sqsubseteq bird \\ penguin \sqsubseteq \neg flies \end{array} \right\}$$

Esta terminología expresa que las aves vuelan y también que los pingüinos son aves pero no son capaces de volar.

Observación 2.3.3 *Nótese que algunas veces nos referiremos a una regla r formada por un axioma de inclusión $C \sqsubseteq D$ o igualdad $C \equiv D$ y la notaremos como $r : C \sqsubseteq D$ o $r : C \equiv D$, respectivamente.*

Tbox $T_{star\text{-}wars}$:

r_1 : $Woman \equiv Person \sqcap Female$
r_2 : $Man \equiv Person \sqcap Male$
r_3 : $Male \equiv \neg Female$
r_4 : $Mother \equiv Woman \sqcap \exists hasChild.\top$
r_5 : $Father \equiv Man \sqcap \exists hasChild.\top$
r_6 : $Son \equiv Male \sqcap \exists hasParent.\top$
r_7 : $Daughter \equiv Female \sqcap \exists hasParent.\top$
r_8 : $Parent \equiv Father \sqcup Mother$
r_9 : $hasParent \equiv hasChild^{-}$

r_{10} : $City \sqcup Planet \sqcup Ship \sqsubseteq Place$
r_{11} : $\{alderaan, bespin, corellia, tattoine\} \sqsubseteq Planet$
r_{12} : $\{deathStar, milleniumFalcon\} \sqsubseteq Ship$
r_{13} : $\{cloudCity\} \sqsubseteq City$
r_{14} : $PlaceForLanding \sqsubseteq Place \sqcap \neg Destroyed$
r_{15} : $Destroyed \sqsupseteq \exists destroyedBy.Weapon$
r_{16} : $homeworldOf \sqcup passengerOn \sqsubseteq knowsAPlace$
r_{17} : $City \sqsubseteq \exists locationOf.Planet$
r_{18} : $commanderOf \equiv commandedBy^{-}$
r_{19} : $Ship \sqsubseteq \exists commandedBy.\top$
r_{20} : $Commander \equiv \exists commanderOf.\top$

Figura 2.2: Ontología $\Sigma_{2.3.7} = (T_{star\text{-}wars}, A_{star\text{-}wars})$ sobre la *Guerra de la Galaxias* (primera parte)

2.3.2. Inferencias

Un sistema de representación de conocimiento basado en las Lógicas para la Descripción es capaz de realizar tipos específicos de razonamiento. El propósito de un sistema

Abox $A_{\textit{star-wars}}$:

a_1 : $leiaOrgana : Person$
a_2 : $leiaOrgana : Female$
a_3 : $lukeSkywalker : Person$
a_4 : $lukeSkywalker : Male$
a_5 : $landoCalrissian : Person$
a_6 : $landoCalrissian : Male$
a_7 : $hanSolo : Person$
a_8 : $hanSolo : Male$
a_9 : $anakinSkywalker : Person$
a_{10} : $anakinSkywalker : Male$
a_{11} : $\langle hanSolo, corellia \rangle : homeworldOf$
a_{12} : $\langle lukeSkywalker, tattoine \rangle : homeworldOf$
a_{13} : $\langle leiaOrgana, alderaan \rangle : homeworldOf$
a_{14} : $\langle anakinSkywalker, tattoine \rangle : homeworldOf$
a_{15} : $\langle anakinSkywalker, lukeSkywalker \rangle : hasChild$
a_{16} : $\langle anakinSkywalker, leiaOrgana \rangle : hasChild$
a_{17} : $\langle lukeSkywalker, milleniumFalcon \rangle : passengerOn$
a_{18} : $\langle leiaOrgana, milleniumFalcon \rangle : passengerOn$
a_{19} : $\langle cloudCity, bespin \rangle : locationOf$
a_{20} : $deathStar : Weapon$
a_{21} : $deathStar : Ship$
a_{22} : $milleniumFalcon : Ship$
a_{23} : $\langle hanSolo, milleniumFalcon \rangle : commanderOf$
a_{24} : $\langle landoCalrissian, milleniumFalcon \rangle : commanderOf$
a_{25} : $\langle alderaan, deathStar \rangle : destroyedBy$

Figura 2.3: Ontología $\Sigma_{2.3.7} = (T_{\textit{star-wars}}, A_{\textit{star-wars}})$ sobre la *Guerra de la Galaxias* (continuación)

de representación de conocimiento va más allá de almacenar descripciones de conceptos y aserciones. Una base de conocimiento (u ontología) tiene una semántica que la hace equivalente a un conjunto de axiomas de la lógica de predicados de primer orden. Así, una ontología contiene conocimiento implícito que puede hacerse explícito a través de inferencias.

Como mencionamos anteriormente, la idea subyaciendo la iniciativa de la Web Semántica es que los programas puedan razonar sobre los datos de la Web. Para ello, el significado de los datos de la Web debe ser definido precisamente. El significado de dichos datos se realiza mediante la definición de ontologías. Cuando las ontologías son consistentes, existen mecanismos de razonamiento eficientes. En cambio, cuando las ontologías son inconsis-

tentes, los mecanismos estándar son incapaces de lidiar con tales inconsistencias.

En esta sección repasaremos las principales tareas de razonamiento en Lógicas para la Descripción, para luego, en el Capítulo 4, estudiar cómo modelar dichas tareas de razonamiento en un marco argumentativo y, particularmente, en presencia de ontologías inconsistentes, para tratar de encontrar conclusiones significativas a partir de tales ontologías inconsistentes.

Tareas de razonamiento para Tboxes

Cuando un ingeniero de conocimiento modela un dominio, construye una terminología T por medio de la definición de conceptos (quizá en términos de conceptos definidos previamente). Durante este proceso es importante determinar si un nuevo concepto tiene sentido o es contradictorio. Desde un punto de vista lógico tradicional, un concepto tiene sentido si existe alguna interpretación que es modelo de T tal que el concepto denota un conjunto no vacío en tal interpretación. Un concepto con tal propiedad se dice *satisfacible* con respecto a T; en caso contrario, se dice *insatisfacible*.

La verificación de la satisfacibilidad de conceptos es una inferencia clave en el marco de las Lógicas para la Descripción. Otros tipos de inferencias pueden ser reducidos a la satisfacibilidad de conceptos. Por ejemplo, el problema de la *subsunción* (*i.e.*, determinar si un concepto está contenido en otro) puede ser resuelto mediante la satisfacibilidad de conceptos.

A continuación presentamos los problemas de razonamiento típicos para terminologías expresadas en Lógicas para la Descripción (Baader et al., 2003) suponiendo que T es una TBox:

- **Satisfacibilidad de conceptos**: Un concepto C es *satisfacible* con respecto a T si y sólo si existe un modelo $\mathcal{I}$ de T tal que $C^{\mathcal{I}}$ es no vacío.
- **Subsunción de clases**: Un concepto C es *subsumido* por un concepto $\mathcal{D}$ con respecto a T si y sólo si $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ para cada modelo $\mathcal{I}$ de T.
- **Equivalencia**: Dos conceptos C y D son *equivalentes* con respecto a T si y sólo si $C^{\mathcal{I}} = D^{\mathcal{I}}$ para cada modelo $\mathcal{I}$ de T.
- **Disyunción**: Dos conceptos C y D son *disjuntos* con respecto a T si y sólo si $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ para cada modelo $\mathcal{I}$ de T.

Ejemplo 2.3.10 *Considere nuevamente la ontología $\Sigma_{2.3.7} = (T_{star\text{-}wars}, A_{star\text{-}wars})$ presentada en el Ejemplo 2.3.7. Por ejemplo, el concepto "Man" es satisfacible. Además, los conceptos "Mother" y "Woman" no son equivalentes porque puede haber mujeres que no sean madres. Por otro lado, los conceptos "Mother" y "Father" son disjuntos, ya que "Male" y "Female" lo son.*

Ejemplo 2.3.11 *Considere nuevamente la ontología $\Sigma_{2.3.8}$ presentada en el Ejemplo 2.3.8. En esta ontología ocurre que el concepto kayak es subsumido por el concepto vehículo, puesto que un kayak es un tipo de bote, que a su vez es un tipo de vehículo acuático, que por su parte es un tipo de vehículo.*

Ejemplo 2.3.12 *Considere nuevamente la terminología $T_{2.3.9}$ presentada en el Ejemplo 2.3.9. En este caso el concepto pingüino es insatisfacible, puesto que como los pingüinos son aves y las aves pueden volar, ocurre que los pingüinos pertenecen a la intersección del concepto voladores y su complemento (el cual es vacío).*

Tradicionalmente, los mecanismos básicos de razonamiento provistos por las Lógicas para la Descripción verificaron la subsunción de conceptos. Esto es de hecho suficiente para implementar las otras inferencias como se puede apreciar en las siguientes reducciones.

Propiedad 2.3.1 (Reducción a subsunción (Baader et al., 2003)) *Para conceptos C, D, tenemos: (1) C es insatisfacible si y sólo si C es subsumido por $\bot$; (2) C y D son equivalentes si y sólo si C es subsumido por D y D es subsumido por C; (3) C y D son disjuntos si y sólo si $C \sqcap D$ es subsumido por $\bot$.*

Además, si el sistema permite representar la negación, los problemas de subsunción, equivalencia y disyunción pueden ser reducidos al problema de la insatisfacibilidad de conceptos.

Propiedad 2.3.2 (Reducción a insatisfacibilidad (Baader et al., 2003)) *Para conceptos C, D, tenemos: (1) C es subsumido por D si y sólo si $C \sqcap \neg D$ es insatisfacible; (2) C y D son equivalentes si y sólo si $(C \sqcap \neg D)$ y $(\neg C \sqcap D)$ son insatisfacibles; (3) C y D son disjuntas si y sólo si $C \sqcap D$ es insatisfacible.*

Propiedad 2.3.3 (Reducción a insatisfacibilidad (Baader et al., 2003)) *Sea C un concepto. Las siguientes afirmaciones son equivalentes: (1) C es insatisfacible; (2) C es subsumido por $\bot$; (3) C y $\bot$ son equivalentes; (4) C y $\top$ son disjuntos.*

Tareas de razonamiento para ABoxes

De acuerdo a Baader et al. (2003, pág. 66), en el marco tradicional de DL, luego de que un ingeniero de conocimiento ha diseñado una terminología y ha usado los servicios de razonamiento de las DL para determinar que todos los conceptos son *satisfacibles* y que se cumplen las relaciones de subsunción esperadas, la Abox puede ser llenada con aserciones sobre individuos (de la forma $a : C$) y relaciones (de la forma $\langle a, b\rangle : R$). Desde el punto de vista de la lógica de primer orden, dicho conocimiento debe ser consistente, porque de otra manera podríamos obtener conclusiones arbitrarias a partir del mismo. Si bien en DL existen varios problemas de inferencia para Aboxes, existe un único problema

de inferencia principal, a saber el chequeo de consistencia para ABoxes, al cual pueden ser reducidos todos los otros problemas de inferencias.

- **Consistencia:** Una Abox A es *consistente con respecto a* una Tbox T si existe una interpretación que es modelo de A y T al mismo tiempo.

- **Chequeo de instancia:** El *chequeo de instancia* consiste en determinar si una aserción se sigue de una Abox. Por ejemplo, $T \cup A \models a : C$ indica que el individuo a pertenece al concepto C con respecto a la Abox A y Tbox T.

- **Recuperación:** Si consideramos a una base de conocimiento como un medio para almacenar información acerca de individuos, podríamos querer conocer todos los individuos que son instancias una descripción de concepto dada. Dada una ontología (T, A) y un concepto C, en el *problema de la recuperación* nos interesa conocer todos los individuos a tales que $T \cup A \models a : C$. Una variante de este problema consiste en hallar todos los individuos que "*llenan*" un rol determinado; formalmente, dado un rol r y un individuo a, hallar todos los individuos b tales que $T \cup A \models \langle a, b \rangle : r$. Una versión no optimizada para el problema de la recuperación puede realizarse verificando para cada individuo de la Abox A si es una instancia del concepto C.

- **Problema de la realización:** La inferencia dual a la recuperación es el *problema de la realización*. Dado un individuo a y un conjunto de conceptos, hallar el concepto más específico C tal que $T \cup A \models a : C$. Aquí, los conceptos más específicos son minimales con respecto al orden inducido por la relación de subsunción $\sqsubseteq$.

Ejemplo 2.3.13 *Considere nuevamente la ontología $\Sigma_{2.3.7} = (T_{star\text{-}wars}, A_{star\text{-}wars})$ presentada en el Ejemplo 2.3.7. Respecto de la operación de consistencia, la Abox $A_{star\text{-}wars}$ es consistente con respecto a la Tbox $T_{star\text{-}wars}$. Respecto de la operación de chequeo de instancia, el individuo Anakin Skywalker es una instancia del concepto "Padre" puesto que es una persona de sexo masculino y tiene dos hijos llamados Leia Organa y Luke Skywalker. Respecto del problema de la recuperación, podemos hallar todas las instancias del concepto "Comandante" y hallar a Han Solo y Lando Calrissian.*

Ejemplo 2.3.14 *Considere nuevamente la ontología $\Sigma_{2.3.8}$ presentada en el Ejemplo 2.3.8. En este caso, el individuo "corsa" es una instancia del concepto "vehículo" puesto que es un auto, los autos son vehículos terrestres y los vehículos terrestres son vehículos. Por otro lado, al recuperar los individuos pertenecientes al concepto "vehículo", hallamos a "boeing747", "corsa" y "scania".*

2.3.3. Relación con la Lógica de Predicados

Como ya hemos comentado en Capítulo 1, en el marco de este trabajo de tesis proponemos resolver los problemas de razonamiento en las Lógicas para la Descripción en un marco argumentativo de tal manera de resolver el problema de la inconsistencia en las ontologías. Debido a que el lenguaje subyacente del lenguaje argumentativo que usaremos está basado en la Programación en Lógica, y como la Programación en Lógica está relacionada con la Lógica de Predicados (a través de la Lógica de Horn) (Lloyd, 1987), es importante conocer la relación de las Lógicas para la Descripción con la Lógica de Predicados de Primer Orden.

La semántica de los conceptos identifica a los lenguajes de descripción como fragmentos de la Lógica de Predicados de Primer Orden de la siguiente forma. Debido a que una interpretación $\mathcal{I}$ asigna a cada concepto una relación unaria sobre $\Delta^{\mathcal{I}}$ y a cada rol una relación sobre $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, podemos ver los conceptos atómicos y roles como predicados unarios y binarios. Entonces, cualquier concepto C puede ser traducido efectivamente en una fórmula de la lógica de predicados $\phi_C(x)$ con una variable libre x tal que, por cada interpretación $\mathcal{I}$, el conjunto de los elementos satisfaciendo $\phi_C(x)$ es exactamente $C^{\mathcal{I}}$.

La traducción se realiza de la siguiente manera: Un concepto atómico A se traduce como la fórmula $A(x)$; es decir, $\phi_A(x) = A(x)$. Los constructores intersección, unión y negación se traducen como una conjunción, disyunción y negación lógicas, respectivamente; es decir:

$$\begin{array}{rcl} \phi_{C \sqcap D}(x) & = & \phi_C(x) \wedge \phi_D(x) \\ \phi_{C \sqcup D}(x) & = & \phi_C(x) \vee \phi_D(x) \\ \phi_{\neg C}(x) & = & \neg \phi_C(x). \end{array}$$

Si C ya se halla traducido como $\phi_C(x)$ y R es un rol atómico, entonces las restricciones de valor y la cuantificación existencial son capturadas por las fórmulas:

$$\begin{array}{rcl} \phi_{\forall R.C}(y) & = & \forall x.R(y,x) \wedge \phi_C(x) \\ \phi_{\exists R.C}(y) & = & \exists x.R(y,x) \rightarrow \phi_C(x) \end{array}$$

donde y es una variable nueva. Las restricciones númericas son expresadas por las fórmulas:

$$\begin{array}{rcl} \phi_{\geq nR}(x) & = & \exists y_1, \ldots, y_n.R(x,y_1) \wedge \ldots \wedge R(x,y_n) \wedge \bigwedge_{i<j} y_i \neq y_j \\ \phi_{\leq nR}(x) & = & \forall y_1, \ldots, y_{n+1}.R(x,y_1) \wedge \ldots \wedge R(x,y_{n+1}) \rightarrow \bigvee_{i<j} y_i = y_j. \end{array}$$

Nótese que el predicado de igualdad "=" es necesario para expresar restricciones numéricas, mientras que los conceptos sin restricciones numéricas pueden ser traducidos como fórmulas sin igualdad.

Por otro lado, los axiomas del tipo *one-of* se representan en lógica de primer orden como:

$$\phi_{\{a_1,\ldots,a_n\}}(x) \quad = \quad (x = a_1) \vee \ldots \vee (x = a_n).$$

Finalmente, los axiomas de inclusión de la forma $C \sqsubseteq D$ pueden ser vistos como una implicación lógica de la forma $\forall x.C(x) \rightarrow D(x)$. Por otro lado, los axiomas de igualdad pueden ser vistos como equivalencias lógicas de la forma $\forall x.C(x) \leftrightarrow D(x)$. La Tabla 2.3 (en la página 46) resume las relaciones de equivalencia entre las Lógicas para la Descripción y la Lógica de Primer Orden.

Cabe comentar que se podría argumentar que, debido a que las Lógicas para la Descripción pueden traducirse a la Lógica de Predicados, no existe la necesidad de una sintaxis especial. Sin embargo, Baader y Nutt (Baader et al., 2003, Cap. 2) argumentan que, en particular, para las restricciones numéricas, la sintaxis libre de variables de las Lógicas para la Descripción es mucho más concisa, lo cual también ayuda al desarrollo de algoritmos de inferencia especializados.

2.3.4. Extensiones para representar dominios concretos

Las Lógicas para la Descripción (Baader et al., 2003) son una familia de formalismos lógicos que incialmente fueron diseñados para la representación de conocimiento conceptual en Inteligencia Artificial y están cercanamente relacionadas a las lógicas modales. En las últimas dos décadas, las Lógicas para la Descripción han sido aplicadas a un gran rango de aplicaciones. En muchas de esas aplicaciones es importante equipar a las Lógicas para la Descripción con medios que permitan describir "cualidades concretas" de objetos del mundo real como su peso, temperatura, etc. El acercamiento estándar (Lutz, 2003) consiste de aumentar las Lógicas para la Descripción con los llamados *dominios concretos*, que consisten de un conjunto (por ejemplo, los números racionales), y un conjunto de predicados n-arios con un extensión fija sobre este conjunto.

Como vimos en la Sección 2.3, la entidad básica de las DLs son los *conceptos*, los cuales son construidos a partir de nombres de conceptos (predicados unarios) y nombres de roles (relaciones binarias) usando el conjunto de constructores de conceptos y roles provistos por la DL particular.

Ejemplo 2.3.15 (Tomado de (Lutz, 2003)) *Por ejemplo, el siguiente concepto es formulado en la DL $\mathcal{ALC}$ para describir un proceso de producción que tiene un operador caro (especialmente entrenado) (Lutz, 2003):*

$$Process \sqcap \forall subproc.Process \sqcap \exists operator.(Human \sqcap Expensive)$$

Como mencionamos más arriba, una extensión importante está dada por los llamados *dominios concretos*, que permiten la integración de "cualidades concretas" como números, intervalos de tiempo y cadenas de caracteres a las descripciones de conceptos. Consideremos un ejemplo presentado por (Lutz, 2003); supongamos que queremos refinar la descripción de un proceso dada más arriba reemplazando el nombre de concepto "*Expensive*" con un concepto expresando que el operador del proceso gana al menos 20 euros por hora.

Entonces es necesario representar números como "20" y comparaciones entre números tales como "al menos 20 euros".

El primer tratamiento formal de dicho problema fue presentado por (Baader and Hanschke, 1991), quienes propusieron extender la DL $\mathcal{ALC}$ con dominios concretos. Formalmente, un dominio concreto consiste de un conjunto como los números naturales y un conjunto de predicados como el binario "$<$" y el ternario "$+$" con una extensión *fija* sobre este conjunto. Al enriquecer $\mathcal{ALC}$ con un tal dominio concreto $\mathcal{D}$, se obtiene la DL $\mathcal{ALC}(\mathcal{D})$. Más precisamente, $\mathcal{ALC}(\mathcal{D})$ es obtenida a partir de $\mathcal{ALC}$ al aumentarla con:

- *atributos abstractos* (en inglés, *abstract features*), es decir, roles interpretados como relaciones funcionales;
- *atributos concretos* (en inglés, *concrete features*): un nuevo tipo sintáctico que es interpretado como una función parcial a partir del dominio lógico en el dominio concreto;
- un nuevo constructor de conceptos que permite describir restricciones sobre valores concretos usando predicados a partir del dominio concreto.

Ejemplo 2.3.16 (Tomado de (Lutz, 2003)) *Retomando el ejemplo anterior, el concepto:*

$$\mathit{Process} \sqcap \forall \mathit{subproc}.\mathit{Process} \sqcap \exists \mathit{operator}.(\mathit{Human} \sqcap \exists \mathit{wage}.\ \geq_{20})$$

refina la descripción de un proceso al reemplazar el nombre de concepto "Expensive" con una descripción basada en un dominio concreto del sueldo del operador, el cual es al menos 20 euros por hora. En este ejemplo, "operator" es un atributo abstracto mientras que "wage" es un atributo concreto. Se utiliza un dominio concreto basado en los números naturales y se asume que "$\geq_{20}$" es un predicado unario con la extensión obvia. El (sub)concepto "$\exists \mathit{wage}.\ \geq_{20}$" es una instanciación del constructor de dominio concreto y no debe ser confundido con la restricción existencial como es usada en "$\exists \mathit{operator}.\mathit{Human}$".

Ejemplo 2.3.17 (Tomado de (Lutz et al., 2004)) *El concepto*

$$\mathit{Employee} \sqcap \exists \mathit{employer}.(\exists \mathit{foundingyear}.\ <_{1970}) \sqcap \exists \mathit{hiringyear}, \mathit{employer}\mathit{foundingyear}.\ \geq$$

describe el conjunto de empleados quienes están empleados por una compañía fundada antes de 1970 y quienes tiene un año de contratación no anterior al año de fundación de la compañía. Aquí, el término dentro de los paréntesis y el tercer miembro de la conjunción son instancias de constructor de dominio concreto, "employer" es un atributo abstracto, y "foundingyear" y "hiringyear" son atributos concretos.

Formalmente, la sintaxis de los conceptos $\mathcal{ALC}(\mathcal{D})$ es definida como sigue:

Definición 2.3.2 (Sintaxis $\mathcal{ALC}(\mathcal{D})$ (Baader and Hanschke, 1991)) *Sean N_C, N_R y N_F conjuntos disjuntos de conceptos, roles y nombres de atributos. El conjunto de conceptos $\mathcal{ALC}(\mathcal{D})$ es el conjunto más pequeño tal que:*

1. *cada nombre de concepto es un concepto y*
2. *si C, D son conceptos, R es un rol o nombre de atributo, $P \in pred(\mathcal{D})$ es un nombre de predicado n-ario, y $u_1, \ldots, u_n$ son cadenas de atributos (una cadena de atributos $u = f_1 \circ \ldots \circ f_m$ es una secuencia de atributos f_i), entonces $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$, $(\exists R.C)$, y $P(u_1, \ldots, u_n)$ son conceptos.*

Los conceptos de la forma $P(u_1, \ldots, u_n)$ son llamados *restricciones de predicados*, y los conceptos de la forma $(\forall R.C)$ (resp. $(\exists R.C)$) son llamados *restricciones de valor universales* (resp. *restricciones de valor existenciales*). La semántica de los conceptos se define de la manera basada en teoría de modelos:

Definición 2.3.3 (Semántica $\mathcal{ALC}(\mathcal{D})$ (Baader and Hanschke, 1991)) *Una interpretación $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consiste de un conjunto $\Delta^{\mathcal{I}}$ disjunto de $dom(\mathcal{D})$, llamado el dominio de $\mathcal{I}$, y una función $\cdot^{\mathcal{I}}$ la cual mapea cada concepto a un subconjunto de $\Delta^{\mathcal{I}}$, cada rol a un subconjunto de $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, y cada nombre de atributo $f \in N_F$ a una función parcial $f^{\mathcal{I}} : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{I}} \cup dom(\mathcal{D})$. Además, $\mathcal{I}$ tiene que satisfacer las siguientes propiedades:*

$$
\begin{aligned}
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} : \text{ existe } e \in \Delta^{\mathcal{I}} \text{ con } (d,e) \in R^{\mathcal{I}} \text{ y } e \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} : \text{ para todo } e \in \Delta^{\mathcal{I}}, \text{ si } (d,e) \in R^{\mathcal{I}}, \text{ entonces } e \in C^{\mathcal{I}}\} \\
P(u_1, \ldots, u_n)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} : (u_1^{\mathcal{I}}(a), \ldots, u_n^{\mathcal{I}}(a)) \in P^{\mathcal{D}}\}
\end{aligned}
$$

Otras extensiones están dadas por las llamadas *funciones de agregación*. Los formalismos básicos de las Lógicas para la Descripción, como $\mathcal{ALC}$ (Baader et al., 2003), no permiten predicados *built-in*, como comparación de números, ni funciones de agregación, como sum, min, max, average, count, que son típicas en los sistemas gestores de bases de datos (Silberschatz et al., 1998). Como mencionamos más arriba, el primer déficit fue resuelto en (Baader and Hanschke, 1991), donde una extensión genérica de $\mathcal{ALC}$ por medio de un *dominio concreto* $\mathcal{D}$ fue propuesta. En dicha extensión, llamada $\mathcal{ALC}(\mathcal{D})$, individuos abstractos (los cuales son descriptos usando $\mathcal{ALC}$) pueden ser relacionados a valores en el *dominio concreto* $\mathcal{D}$ (*e.g.*, los enteros, cadenas de caracteres, etc.) a través de los denominados *atributos* (en inglés, *features*), *i.e.*, roles funcionales. Esto permite describir, por ejemplo, los gerentes que gastan más dinero del que ganan por medio de:

$$Manager \sqcap (less(income, expenses)).$$

Cuadro 2.1: Sentencias OWL y axiomas $\mathcal{SHOIQ}$

Axioma	*Sintaxis DL*	*Ejemplo*
`subClassOf`	$C_1 \sqsubseteq C_2$	$Bird \sqsubseteq Animal \sqcap Flies$
`sameClassAs`	$C_1 \equiv C_2$	$Man \equiv Human \sqcap Male$
`subPropertyOf`	$P_1 \sqsubseteq P_2$	$hasBought \sqsubseteq hasAcquired$
`samePropertyAs`	$P_1 \equiv P_2$	$salary \equiv income$
`disjointWith`	$C_1 \sqsubseteq \neg C_2$	$Penguin \sqsubseteq \neg Flies$
`sameIndividualAs`	$\{i_1\} \equiv \{i_2\}$	$\{little\_grasshopper\} \equiv \{quan\_chang\_kein\}$
`differenIndividualFrom`	$\{i_1\} \sqsubseteq \neg\{i_2\}$	$\{master\_kan\} \sqsubseteq \neg\{quan\_chang\_kein\}$
`inverseOf`	$P_1 \equiv P_2^{-1}$	$isTeacherOf \equiv isApprenticeOf^{-1}$
`transitiveProperty`	$P^+ \sqsubseteq P$	$ancestor^+ \sqsubseteq ancestor$
`uniqueProperty`	$\top \sqsubseteq \leq 1P.\top$	$\top \sqsubseteq \leq 1hasMother.\top$
`unambiguousProperty`	$\top \sqsubseteq \leq 1P^-.\top$	$\top \sqsubseteq \leq 1isMotherOf^-.\top$
`range`	$\top \sqsubseteq \forall P.C$	$\top \sqsubseteq \forall commandedBy.Commander$
`domain`	$\top \sqsubseteq \forall P^-.C$	$\top \sqsubseteq \forall commanderOf.Ship$
i `type` C	$i : C$	$milleniumFalcon : Ship$
i_1 `P` i_2	$\langle i_1, i_2 \rangle : P$	$\langle hanSolo, milleniumFalcon \rangle : commandedBy$

Por otro lado, (Baader and Sattler, 1998) presentan otra extensión de $\mathcal{ALC}$ donde se considera la representación de agregaciones como medio para definir nuevos atributos. Por ejemplo, si un empleado que tiene gastos e ingresos, algunos meses gasta más de lo que gana pero en otros ocurre lo contrario (es decir gana más de lo que gasta), la extensión presentada permite determinar la diferencia entre ganancias y gastos a lo largo de un año, de tal manera de poder determinar si un individuo pertenece al concepto:

$$Human \sqcap (\exists year.less(\mathsf{sum}(month \circ income), \mathsf{sum}(month \circ expenses))).$$

donde el atributo complejo "$\mathsf{sum}(month \circ income)$" relaciona un individuo con la suma sobre todos los valores alcanzables sobre "*month*" seguidos de "*income*". Este nuevo atributo complejo es construido utilizando la función de agregación "sum", el nombre de rol "*month*" y el atributo "*income*". Sin embargo, (Baader and Sattler, 1998) aclaran que dados un dominio concreto satisfaciendo condiciones muy débiles, esta extensión tiene un problema de satisfacibilidad y subsunción indecidible.

2.3.5. Relación con lenguajes de representación en la Web

La relevancia de las Lógicas para la Descripción en el marco de la iniciativa de la Web Semántica está dada porque las mismas sirven de marco teórico para dar semántica a los lenguajes de representación de conocimiento en la Web Semántica. En particular el lenguaje OWL-DL tiene como semántica a la Lógica para la Descripción $\mathcal{SHOIQ}$. Las Tablas 2.2 y 2.1 resumen las equivalencias entre OWL y la Lógica DL $\mathcal{SHOIQ}$.

Cuadro 2.2: Constructores de clase OWL

Constructor	*Sintaxis DL*	*Ejemplo*
`intersectionOf`	$C_1 \sqcap \ldots \sqcap C_n$	$Bird \sqcap Flies$
`unionOf`	$C_1 \sqcup \ldots \sqcup C_n$	$FreeDiver \sqcup ScubaDiver$
`complementOf`	$\neg C$	$\neg Flies$
`oneOf`	$\{i_1, \ldots, i_n\}$	$\{lukeSkywalker, joda, obiWan\}$
`hasClass`	$\exists P.C$	$\exists hasChild.Padawan$
`toClass`	$\forall P.C$	$\forall hasChild.Padawan$
`hasValue`	$\exists P.\{i\}$	$\exists isTeacherOf.\{lukeSkywalker\}$
`minCardinalityQ`	$\geq nP.C$	$\geq 2 hasChild.Padawan$
`maxCardinalityQ`	$\leq nP.C$	$\leq 1 hasChild.Padawan$
`cardinalityQ`	$= nP.C$	$= 1 isTeacherOf.Padawan$

Cuadro 2.3: Equivalencia entre las Lógicas para la Descripción y la Lógica de Primer Orden

Lógicas para la Descripción	*Lógica de Primer Orden*
$a : C$	$C(a)$
$\langle a, b \rangle : P$	$P(a, b)$
$C \sqsubseteq D$	$(\forall x)(C(x) \rightarrow D(x))$
$P^+ \sqsubseteq P$	$(\forall x, y, z)((P(x, y) \wedge P(y, z)) \rightarrow P(x, z))$
$\top \sqsubseteq \leq 1P$	$(\forall x, y, z)((P(x, y) \wedge P(y, z)) \rightarrow (x = z))$
$P \equiv Q^-$	$(\forall x, y)(P(x, y) \leftrightarrow Q(y, x))$
$C_1 \sqcap \ldots \sqcap C_n$	$C_1(x) \wedge \ldots \wedge C_n(x)$
$C_1 \sqcup \ldots \sqcup C_n$	$C_1(x) \vee \ldots \vee C_n(x)$
$\neg C$	$\neg C(x)$
$\{a_1, \ldots, a_n\}$	$(x = a_1) \vee \ldots \vee (x = a_n)$
$\exists P.C$	$(\exists y)(P(x, y) \wedge C(y))$
$\forall P.C$	$(\forall y)(P(x, y) \rightarrow C(y))$
$\geq nP.C$	$(\exists y_1, \ldots, y_n)(\bigwedge_{1 \leq i \leq n}(P(x, y_i) \wedge C(y_i)) \wedge \bigwedge_{1 \leq i < n, i < j \leq n}(y_i \neq y_j))$
$\leq (n-1)P.C$	$(\forall y_1, \ldots, y_n)(\bigwedge_{1 \leq i \leq n}(P(x, y_i) \wedge C(y_i)) \rightarrow \bigvee_{1 \leq i < n, i < j \leq n}(y_i = y_j))$

2.4. Inconsistencia en Lógicas para la Descripción

Las ontologías juegan un rol crucial en la Web Semántica, ya que permiten que agentes inteligentes compartan información en una forma semánticamente no ambigua y reusen conocimiento del dominio de aplicación posiblemente creado por fuentes externas (Schlobach and Huang, 2005). Sin embargo, esta situación hace que la tecnología de la Web Semántica sea altamente dependiente de la calidad y, en particular, de la correctitud de la ontología utilizada. Una manera de conseguir la correctitud de una ontología consiste en la utilización de sofisticadas herramientas de modelado de ontologías, otra manera está basada en el razonamiento lógico.

Un caso de incorrectitud de una ontología está dado por la existencia de incoherencias en la forma de inconsistencias. La inconsistencia en ontologías DL es definida como sigue:

Definición 2.4.1 (Inconsistencia en ontologías (Baader et al., 2003)) *Sea $\vdash_{DL}$ la relación de inferencia en lógicas para la descripción. Una ontología $\Sigma = (T, A)$ es inconsistente, denotado como $\Sigma \vdash_{DL} \top \subseteq \bot$ si para todas las interpretaciones $\mathcal{I}$ de Σ, la interpretación de Σ es un modelo vacío.*

Con el advenimiento de lenguajes de ontologías como OWL y su cercana relación con las Lógicas para la Descripción, información implícita no-trivial, como la jerarquía de clases puede ser explicitada por los razonadores lógicos. Sin embargo, si bien los razonadores modernos para las Lógicas para la Descripción (*e.g.*, Racer (Haarslev and Möller, 2001, 2004)) pueden detectar inconsistencias en ontologías extensas, el problema de la solución de las mismas sigue presente por varios motivos. En muchos casos, es muy difícil para el ingeniero de conocimiento eliminar las incoherencias por varios motivos: (i) el dominio modelado puede ser muy complejo; (ii) varios expertos en el dominio modelado pueden estar en desacuerdo. Por otro lado, en el caso de ontologías importadas de fuentes externas, el ingeniero de conocimiento puede no tener autoridad para corregir tales ontologías importadas.

En el resto de esta sección, enumeraremos las causas de la presencia de inconsistencias en ontologías y consideraremos el comportamiento de los razonadores estándar frente a la presencia de incoherencias e inconsistencias en ontologías expresadas en Lógicas para la Descripción.

2.4.1. Causas de la inconsistencia

Hay varios escenarios típicos que pueden causar inconsistencia en la Web Semántica (Huang et al., 2004):

- inconsistencia por presentación incorrecta de reglas;
- inconsistencia causada por polisemia;

- inconsistencia causada por migración de otro formalismo, e,
- inconsistencia causada por múltiples fuentes.

A continuación presentamos en detalle cada uno de estos escenarios.

Inconsistencia por presentación incorrecta de reglas: Cuando un ingeniero de conocimiento especifica una sentencia de una ontología, en particular una regla, tiene que verificar cuidadosamente que la nueva sentencia sea consistente, no sólo con respecto a las reglas existentes sino también con respecto a las reglas que puedan ser agregadas en el futuro, las cuáles por supuesto no son conocidas siempre en el momento. Esto hace muy difícil el mantenimiento de la consistencia en las especificación de ontologías.

Ejemplo 2.4.1 (Adaptado de (Huang et al., 2004)) *Consideremos una situación en la cual un ingeniero de conocimiento quiere crear una ontología sobre animales:*

$$r_1: \quad \mathit{bird} \sqsubseteq \mathit{animal}$$
$$r_2: \quad \mathit{bird} \sqsubseteq \mathit{fly}$$

La regla r_1 dice que las aves son animales mientras que la regla r_2 establece que las aves vuelan. A pesar de que el ingeniero en conocimiento entiende que la sentencia "las aves vuelan" no es generalmente válida, aún así desea agregarla si no halla ningún contraejemplo en la base de conocimiento corriente, ya que volar es una de las cualidades principales de las aves.

Luego, uno puede desear agregar las siguientes sentencias:

$$r_3: \quad \mathit{eagle} \sqsubseteq \mathit{bird}$$
$$r_4: \quad \mathit{penguin} \sqsubseteq \mathit{bird}$$
$$r_5: \quad \mathit{penguin} \sqsubseteq \neg \mathit{fly}$$

La regla r_3 establece que las águilas son aves, la regla r_4 dice que los pingüinos son aves, y la regla r_5 especifica que los pingüinos no vuelan). El concepto penguin en dicha ontología sobre aves es insatisfacible, pues implica que los pingüinos pueden volar y no volar al mismo tiempo. Esto da lugar a una ontología inconsistente.

*Una solución sería eliminar la regla "las aves vuelan" (*i.e., *la regla r_2) de la ontología para restaurar la consistencia. Sin embargo, este enfoque no es viable por las siguientes razones: (a) es difícil verificar que la remoción no causa ninguna pérdida significativa de información en la ontología actual; (b) puede ser que no tengamos autoridad para remover sentencias pertenecientes a la ontología actual, y (c) puede ser difícil determinar qué parte de la ontología actual puede ser removida en el caso en que ésta sea muy extensa.*

En el ejemplo anterior, el problema yace en que los actuales lenguajes de representación de conocimiento en la Web Semántica son incapaces de lidiar con estos problemas (McGuiness and van Harmelen, 2004) debido a que están basados en reglas de inferencia

monótonas. Algunos autores plantean como una solución particular la de partir la ontología en dos o más ontologías donde cada parte es consistente (Huang et al., 2004).

Ejemplo 2.4.2 (Adaptado de (de Bruijn, 2005)) *Considere la siguiente terminología $T_{2.4.2}$ sobre la enfermedad de las vacas locas tal que:*

$$T_{2.4.2} = \left\{ \begin{array}{l} Cow \equiv Animal \sqcap Vegetarian \\ Sheep \sqsubseteq Animal \\ Vegetarian \equiv \forall eats.\neg Animal \\ MadCow \equiv Cow \sqcap \exists eats.Sheep \end{array} \right\}.$$

La primera regla indica que las vacas son animales vegetarianos; la segunda, que las ovejas son animales; la tercera que los vegetarianos no comen animales, y, la cuarta, que las vacas locas son vacas que comen ovejas. Claramente el concepto "MadCow" es inconsistente ya que una vaca loca come ovejas, que son animales, cuando no debiera ya que las vacas locas son vacas, que, por definición, son animales vegetarianos. [8]

Ejemplo 2.4.3 (Tomado de (Antoniou and van Harmelen, 2003)) *Supongamos que hemos declarado a x como instancia de una clase A. Además, supongamos que:*

- *A es una subclase de $B \cap C$*
- *A es una subclase de D*
- *B y D son disjuntas.*

Entonces, tenemos una inconsistencia porque A debería ser vacía, pero tiene la instancia x. Esto es una indicación de un error en la ontología.

Inconsistencia causada por polisemia: La *polisemia* se refiere al concepto de palabras con significados múltiples. Cuando una ontología es especificada, se debe tener un entendimiento claro de todos los conceptos descriptos.

Ejemplo 2.4.4 (Adaptado de (Huang et al., 2004)) *Consideremos la siguiente terminología:*

$$\begin{array}{ll} r_1: & married Woman \sqsubseteq woman \\ r_2: & married Woman \sqsubseteq \neg divorcee \\ r_3: & divorcee \sqsubseteq had Husband \sqcap \neg has Husband \\ r_4: & has Husband \sqsubseteq married Woman \\ r_5: & had Husband \sqsubseteq married Woman \end{array}$$

[8] En (de Bruijn, 2005, Slide 30), el tercer axioma es en realidad presentado como "$Vegetarian \equiv \exists eats.\neg Animal$". Creemos que dicha formalización es incorrecta, ya que la definición de "*vegetarianismo*" implica que un vegetariano no come carne. Es de notar que en la versión original de De Bruijn el concepto "*MadCow*" es consistente.

La regla r_1 especifica que las mujeres casadas son mujeres, la regla r_2 dice que las mujeres casadas no son divorciadas, la regla r_3 establece que las divorciadas no tienen marido pero sí tuvieron marido, la regla r_4 especifica que alguien que tiene marido significa que alguna vez se casó, y la regla r_5 dice que alguien tuvo marido significa que se casó en algún momento.

En la especificación de la ontología anterior, el concepto "*divorcee*" es insatisfacible debido al uso incorrecto del término "*marriedWoman*". Por lo tanto, es necesario verificar si existe algún malentendido con respecto a los conceptos que han sido usados en la ontología. Nuevamente, cuando la ontología es extensa, este tipo de requerimiento se vuelve difícil de satisfacer.

Inconsistencia a través de migración desde otro formalismo: Cuando la especificación de una ontología es migrada desde otra fuente de datos, también pueden ocurrir inconsistencias. Por ejemplo, la ontología DICE (*Diagnoses for Intensive Care Evaluation* o *Diagnósticos para Evaluación de Terapia Intensiva*) [9] está basada en un sistema de marcos (*frames*) (Minsky, 1975). Para actualizarla a los formatos actuales, DICE fue traducida a DL. En tal proceso, como puntualiza (Huang et al., 2004) citando a (Schlobach and Cornet, 2003), aparecieron un alto número de conceptos insatisfacibles en la terminología DL resultante para la original de DICE.

Ejemplo 2.4.5 (Adaptado de (Huang et al., 2004)) *Por ejemplo, consideremos la siguiente especificación médica inconsistente:*[10]

$$
\begin{aligned}
r_1 &: \ \mathit{brain} \sqsubseteq \mathit{centralNervousSystem} \\
r_2 &: \ \mathit{brain} \sqsubseteq \mathit{bodyPart} \\
r_3 &: \ \mathit{centralNervousSystem} \sqsubseteq \mathit{nervousSystem} \\
r_4 &: \ \mathit{bodyPart} \sqsubseteq \neg \mathit{nervousSystem}
\end{aligned}
$$

La regla r_1 establece que un cerebro es parte del sistema nervioso central, la regla r_2 dice que el cerebro es una parte del cuerpo, la regla r_3 especifica que el sistema nervioso central es parte del sistema nervioso, la regla r_4 establece que una parte del cuerpo no es parte del sistema nervioso. Claramente el concepto cerebro es insatisfacible ya que por un lado está contenido en el sistema nervioso central (por la regla r_1) que a su vez está contenido en el sistema nervioso (por la regla r_3). Por otro lado, el cerebro es un parte del cuerpo (por la regla r_2) que a su vez no es parte del sistema nervioso (por la regla r_4). Así el cerebro es y no es al mismo tiempo parte del sistema nervioso. Por lo tanto, el concepto cerebro está subsumido por el concepto vacío (es decir, "$\mathit{brain} \sqsubseteq \bot$").

[9] http://kik.amc.uva.nl/dice/

[10] Esta ontología también es mencionada en (Meyer et al., 2005).

Inconsistencia causada por múltiples fuentes: Cuando una especificación ontológica es generada a partir de fuentes múltiples, en particular cuando estas fuentes son creadas por autores múltiples, fácilmente pueden ocurrir inconsistencias. El siguiente ejemplo ilustra la situación:

Ejemplo 2.4.6 *Dadas dos ontologías* $\Sigma_1 = (T_1, A_1)$ *y* $\Sigma_2 = (T_2, A_2)$ *tales que:*

$$\begin{array}{rcl} T_1 & = & \{\ PhDStudent \sqsubseteq \neg Candidate\ \} \\ A_1 & = & \{\ john : PhDStudent\ \} \\ T_2 & = & \{\ PhDStudent \sqcap Rich \sqsubseteq Candidate\ \} \\ A_2 & = & \{\ john : Rich\ \} \end{array}$$

La terminología T_1 *establece que los estudiantes de doctorado no son candidatos a obtener un préstamo (ya que son pobres). La caja asercional* A_1 *establece que John es un estudiante de doctorado. Por otro lado, la terminología* T_2 *dice que los estudiantes de doctorados que además son ricos sí son candidatos a obtener un préstamo. Además, la caja asercional* A_2 *indica que John es rico. Asumiendo nombres únicos en las dos ontologías, si consideramos la unión irrestricta de las mismas de tal manera de formar* $\Sigma_3 = (T_1 \cup T_2, A_1 \cup A_2)$*, esta nueva ontología es inconsistente ya que permite inferir que "*$john : Candidate$*" y "*$\neg(john : Candidate)$*".*

2.4.2. Los razonadores estándar frente a la inconsistencia

En esta sección, estudiaremos brevemente el comportamiento de un razonador estándar sobre Lógicas para la Descripción, a saber RACER (Haarslev and Möller, 2001, 2004). RACER es un sistema que es posible utilizar mediante una interfaz Java (corriendo como un servidor) o simplemente desde la línea de comando. Cuando se utiliza esta última opción es necesario brindar un archivo de texto con el código fuente de la ontología en cuestión y otro archivo de texto con las consultas a realizar sobre tal ontología. Así, el sistema genera un archivo de texto conteniendo las respuestas a las consultas realizadas.

Veamos como una simple ontología puede presentar inconsistencias.

Ejemplo 2.4.7 *Consideremos la siguiente ontología* $\Sigma_{2.4.7} = (T, A)$ *donde:*

$$\begin{array}{rcl} T & = & \left\{ \begin{array}{l} bird \sqsubseteq flies; \\ penguin \sqsubseteq bird; \\ penguin \sqsubseteq \neg flies \end{array} \right\}, \text{ y} \\ A & = & \left\{ \begin{array}{l} tweety : bird; \\ opus : penguin \end{array} \right\}. \end{array}$$

Al observar la Tbox T vemos que el concepto *penguin* debe ser vacío pero hay un pingüino llamado Opus. Para demostrar que la ontología $\Sigma_{2.4.7}$ es inconsistente es suficiente mostrar que hay un concepto que es vacío pero sin embargo se ha especificado que al

menos un individuo pertenece al mismo. Veamos que el concepto "*penguin*" debe ser vacío. Para mostrar que el concepto "*penguin*" es vacío alcanza con determinar que el mismo es subsumido por el concepto inconsistente: Sabemos que "$bird \sqsubseteq flies$", "$penguin \sqsubseteq bird$" y "$penguin \sqsubseteq \neg flies$" pertenecen a la Tbox T. A partir de las dos primeras sentencias podemos determinar que los pingüinos deben volar por ser aves (*i.e.*, "$penguin \sqsubseteq flies$"); sin embargo, sabemos también por la tercera sentencia de T que los pingüinos no vuelan (*i.e.*, "$penguin \sqsubseteq \neg flies$"). Así, tenemos que "$penguin \sqsubseteq flies \sqcap \neg flies$". Por lo tanto, "$penguin \sqsubseteq \bot$". Pero como "$opus : penguin$" pertenece a la Abox A (*i.e.*, Opus es un pingüino), tenemos una contradicción ya que el conjunto de pingüinos debiera ser vacío y sin embargo hay un pingüino llamado Opus.

Cuando estamos frente a tal ontología, los razonadores estándar para Lógicas para la Descripción al ser alimentados con dicha ontología producen como salida exactamente lo visto en el ejemplo anterior. En la Figura 2.4 (ver página 53), mostramos el código Racer para representar la ontología presentada previamente en el Ejemplo 2.4.7 (véase la página 51).

Cuando alimentamos a un razonador como Racer con esta ontología[11] y con la consulta "$penguin(opus)$?", codificada en Racer como:

```
(individual-instance? opus penguin),
```

el mismo produce la salida:

```
Error: ABox DEFAULT is incoherent.
```

indicando que la ontología es inconsistente. En el caso de eliminar las sentencias correspondientes a la especificación de la Abox, Racer provee la sentencia:

```
(concept-satisfiable? penguin)
```

que nos permitirá determinar que el concepto "Pingüino" no es satisfacible.

Supongamos por un momento que no estamos interesados en determinar si los pingüinos son capaces de volar o no. El problema del enfoque anterior es que respecto de las aves estándar (como Tweety) sí nos interesa poder contestar si es el caso que el mismo es capaz de volar o no. Claramente, dicha situación no ocurre con los razonadores tradicionales.

2.5. Integración de ontologías

2.5.1. Definiciones preliminares

La combinación de dos o más ontologías en una única ontología se conoce usualmente como *integración*. Sin embargo, la terminología respecto el campo de integración de

[11] La consulta se escribe como `racer.exe -f onto-poblada1.racer -q q.racer -o out.txt`.

```
(signature
        :atomic-concepts (bird penguin flies)
        :individuals (tweety opus)
)
(implies bird flies)
(implies penguin bird)
(implies penguin (not flies))

(instance opus penguin)
(instance tweety bird)
```

Figura 2.4: Código RACER para la ontología inconsistente del Ejemplo 2.4.7

ontologías es muy dispar y algunas veces contradictorio ya que varios autores proponen clasificaciones diferentes de la terminología (*e.g.*, (Klein, 2001) y (Pinto et al., 1999; Pinto and Martins, 2001)). En esta sección, revisamos brevemente la terminología asociada con el subcampo de la integración de ontologías.

La *combinación* (en inglés, *combining*) se refiere a usar dos o más ontologías diferentes para una tarea en la que su relación es importante (Klein, 2001); sin embargo, otros autores ven a esta noción como sólo *usar* (en inglés, *using*) ontologías (Pinto et al., 1999); *i.e.*, la integración de ontologías en aplicaciones. La *mezcla/integración* (en inglés, *merging/integration*) es el proceso de crear una nueva ontología a partir de dos o más ontologías existentes con partes solapadas (Klein, 2001). Pinto *et al.* (Pinto et al., 1999) distinguen entre *integración* de ontologías (cuando se construye una nueva ontología reutilizando otras ontologías disponibles) y *mezcla* de ontologías diferentes acerca de un mismo tema en una única ontología que las unifique.

Las ontologías integradas pueden estar en desacuerdo. Por lo tanto, la *alineación* (en inglés, *aligning*) es el proceso de acuerdo entre sí a dos o más ontologías, haciéndolas consistentes entre sí (Klein, 2001). Una *articulación* es el punto de enlace entre dos ontologías alineadas (Klein, 2001). Los puntos de articulación pueden tener semántica entre equivalente, subsunción (`is-a`), propiedad (`part-of` y/o `has-knowledge-of`) (Mitra et al., 2000; Mitra, 2004).

La traducción (en inglés, *translating*) consiste en cambiar el formalismo de representación de una ontología mientras se preserva la semántica. A este respecto, más adelante en esta Tesis, veremos cómo considerar ontologías inconsistentes representadas en un subconjunto de las Lógicas para la Descripción y traducirlas a un programa DeLP para razonar en términos de un programa DeLP.

Los cambios a una ontología resultan en la producción de otra ontología. La *transformación* (en inglés, *transforming*) consiste del cambio pequeño de la semántica de una ontología para hacerla apropiada para un propósito diferente del original. Una *versión* es el resultado de un cambio a una ontología. *Versionado* (en inglés, *versioning*) es un mecanismo para el mantenimiento de la historia de cambios entre las ontologías antiguas y las nuevas ontologías evolucionadas.

La teoría de integración de ontologías derivó de la teoría de integración de datos. La *integración de datos* es el problema de combinar datos residiendo en diferentes fuentes y proveer al usuario con una vista unificada de dichos datos (Lenzerini, 2002). Un tipo de sistemas de integración de datos son aquellos que se caracterizan por una arquitectura basada en un esquema global y un conjunto de fuentes. Las fuentes contienen los datos reales mientras que el esquema global provee una vista virtual, integrada y reconciliada de las fuentes subyacentes. Hay dos acercamientos básicos para implementar la integración de datos: de *arriba hacia abajo* (o *global-as-view* o *centrado globalmente*) y de *de abajo hacia arriba* (o *local-as-view* o *centrado localmente*). Un caso particular se da cuando el esquema global y las fuentes son especificados como *ontologías*, en dicho caso tenemos la llamada *integración de ontologías* (Calvanese et al., 2001).

En el acercamiento *global-as-view* se requiere que el esquema global esté expresado en función de las fuentes de datos. En el acercamiento *local-as-view* requiere que el esquema global se halle especificado independientemente de las fuentes, y las relaciones entre el esquema global y las fuentes se establezca definiendo cada fuente como una vista sobre el esquema global.

Independientemente del método usado para la especificación del mapeo entre el esquema global y las fuentes, un servicio básico provisto por el sistema de integración es el de contestar consultas realizadas con respecto al esquema global. Dada la arquitectura del sistema, el procesamiento de consultas en la integración de datos requiere un paso de reformulación: la consulta sobre el esquema global tiene que ser reformulada en términos de un conjunto de consultas sobre las fuentes.

Debido a que las fuentes son en general autónomas, en muchos escenarios surge el problema de fuentes mutuamente inconsistentes. De acuerdo a Lenzerini (2002), este problema es usualmente resuelto por medio de transformaciones adecuadas y operaciones de limpieza. En esta tesis, consideraremos el problema de la integración de ontologías posiblemente inconsistentes; sin embargo, en lugar de limpiar las ontologías inconsistentes, lidiaremos directamente con las inconsistencias por medio de un análisis dialéctico.

Ahora presentamos el marco abstracto de integración de ontologías presentado por Calvanese (Calvanese et al., 2001), el cual, a su vez, está basado en el marco de integración de datos presentado en (Lenzerini, 2002); el mismo se halla basado en un esquema global o *esquema mediado*, y se refiere a sistemas de integración de datos cuyo propósito es el de combinar los datos residiendo en distintas fuentes y el de proveer al usuario una vista

unificada de dichos datos. Tal vista unificada es representada por el esquema global, y provee una vista reconciliada de todos los datos, los cuales pueden ser consultados por el usuario. El enlace entre el esquema global y las fuentes se realiza por medio de una *correspondencia* entre el esquema global y las fuentes.

Luego, las componentes principales de un sistema de integración de datos son el esquema global, las fuentes y el mapeo. Así, Calvanese (Calvanese et al., 2001) formaliza un *sistema de integración de ontologías* $\mathcal{I}$ en términos de una tripla $\langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$, donde:

- $\mathcal{G}$ es la *ontología global*, expresada como una teoría en una lógica $\mathcal{L}_{\mathcal{G}}$ sobre un alfabeto de términos $\mathcal{A}_{\mathcal{G}}$. El alfabeto contiene un símbolo por cada elemento de $\mathcal{G}$ (es decir, relación si $\mathcal{G}$ es relacional, clase si $\mathcal{G}$ es orientado a objetos, concepto si $\mathcal{G}$ es DL, etc.).

- $\mathcal{S}$ es un conjunto de n *ontologías locales* $\mathcal{S}_1, \ldots, \mathcal{S}_n$. Se denota con $\mathcal{A}_{\mathcal{S}_i}$ el alfabeto de los términos de la ontología local $\mathcal{S}_i$. También se denota con $\mathcal{A}_{\mathcal{S}}$ a la unión de todos los $\mathcal{A}_{\mathcal{S}_i}$. En (Calvanese et al., 2001), se asume que los conjuntos $\mathcal{S}_i$ son mutuamente disjuntos y que cada uno es disjunto con $\mathcal{A}_{\mathcal{G}}$; en la Sección 4.6 veremos que esta restricción no es necesaria si calificamos a cada término con el nombre de la ontología que lo define. Se asume también que cada ontología local está representada con una teoría (llamada $\mathcal{S}_i$) en alguna lógica $\mathcal{L}_{\mathcal{S}_i}$ y se usa $\mathcal{S}$ para denotar a la colección de teorías $\mathcal{S}_1, \ldots, \mathcal{S}_n$.

- $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ es la *correspondencia* entre $\mathcal{G}$ y $\mathcal{S}$, que especifica cómo los conceptos de la ontología global $\mathcal{G}$ y las ontologías locales $\mathcal{S}$ se relacionan entre sí. La correspondencia $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ está constituida por un conjunto de *aserciones* de la forma:

$$q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}}$$
$$q_{\mathcal{G}} \rightsquigarrow q_{\mathcal{S}}$$

  donde $q_{\mathcal{S}}$ son dos consultas de la misma aridad, respectivamente sobre el esquema fuente $\mathcal{S}$ y sobre el esquema global $\mathcal{G}$. Las consultas $q_{\mathcal{S}}$ son expresadas en un lenguaje de consultas $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ sobre el alfabeto $\mathcal{A}_{\mathcal{S}}$, y las consultas $q_{\mathcal{G}}$ son expresadas en un lenguaje de consultas $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ sobre el alfabeto $\mathcal{A}_{\mathcal{G}}$. Intuitivamente, una aserción $q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}}$ especifica que el concepto representado por la consulta $q_{\mathcal{S}}$ sobre las fuentes corresponde al concepto en el esquema global representado por la consulta $q_{\mathcal{G}}$ (respectivamente para una aserción de tipo $q_{\mathcal{G}} \rightsquigarrow q_{\mathcal{S}}$).

El tema de la integración de ontologías se halla relacionado con el tema de *reescritura de conceptos* (en inglés, *Concept Rewriting*) (Nardi and Brachman, 2003, Sección 1.6.2.3). Dado un concepto, expresado en un lenguaje fuente, la reescritura de conceptos se refiere a hallar un concepto, posiblemente expresado en un lenguaje destino, el cual se halla

relacionado con el concepto dado de acuerdo a equivalencia, subsunción o alguna otra relación.

Para especificar la semántica de un sistema de integración de ontologías, se debe comenzar con un modelo de las ontologías locales y luego especificar cuáles son los modelos de la ontología global. Así para asignar semántica a un sistema de integración $\mathcal{O} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$, se considera un modelo local $\mathcal{D}$ para $\mathcal{O}$ (*i.e.*, una interpretación que es modelo para todas las teorías de $\mathcal{S}$). Se llama *interpretación global* para $\mathcal{O}$ a cualquier interpretación para $\mathcal{G}$. Una interpretación global $\mathcal{I}$ para $\mathcal{O}$ se dice un *modelo global* para $\mathcal{O}$ con respecto a $\mathcal{D}$ si: $\mathcal{I}$ es un modelo de $\mathcal{G}$ e $\mathcal{I}$ satisface el mapeo $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ con respecto a $\mathcal{D}$. La semántica de $\mathcal{O}$, denotada *sem*, se define como:

$$\begin{aligned} sem \quad = \{\mathcal{I} \mid \quad & \text{existe un modelo local } \mathcal{D} \text{ para } \mathcal{O} \\ & \text{tal que } \mathcal{I} \text{ es un modelo global para } \mathcal{O} \text{ con respecto a } \mathcal{D}\} \end{aligned}$$

Las consultas realizadas a un sistema de integración de ontologías son expresadas en términos de un lenguaje de consultas $\mathcal{Q}_\mathcal{G}$ sobre el alfabeto $\mathcal{A}_\mathcal{G}$ y tienen como objetivo extraer un conjunto de tuplas de elementos de Δ. Así, cada consulta tiene asociada una aridad, donde la semántica de una consulta q de aridad n se define como: la respuesta $q^\mathcal{O}$ de q con respecto a $\mathcal{O}$ es el conjunto de tuplas

$$q^\mathcal{O} \quad = \left\{ \ \langle c_1, \ldots, c_n \rangle | \text{ para todo } \mathcal{I} \in sem, \langle c_1, \ldots, c_n \rangle \in q^\mathcal{I} \ \right\}$$

donde $q^\mathcal{I}$ denota el resultado de evaluar a q en la interpretación $\mathcal{I}$.

Intuitivamente, los esquemas locales describen la estructura de las fuentes, donde se hallan los datos reales, mientras que el esquema global provee una vista reconciliada, integral y virtual de los datos de las fuentes subyacentes. Las aserciones en las correspondencias establecen la conexión entre los elementos del esquema global y aquéllos de los esquemas fuente. Hay dos acercamientos principales para caracterizar tal correspondencia:

- El acercamiento *de arriba hacia abajo* (o *global-as-view* o *global-centric*), donde los conceptos de la ontología global $\mathcal{G}$ son mapeados en consultas sobre las ontologías locales en $\mathcal{S}$.

- El acercamiento *de abajo hacia arriba* (o *local-as-view* o *local-centric*), donde los conceptos de las ontologías locales en $\mathcal{S}$ son mapeados en consultas sobre la ontología global $\mathcal{G}$.

En las dos siguientes secciones, exploramos estos dos acercamientos con más detenimiento.

2.5.2. Integración de arriba hacia abajo o *global-as-view*

En el acercamiento *global-as-view* (GAV) a la integración de ontologías, se asume que se tiene un lenguaje $\mathcal{V}_\mathcal{S}$ sobre el alfabeto $\mathcal{A}_\mathcal{S}$, y la correspondencia entre la ontología global y las ontologías locales es dada al asociar a cada término de la ontología global una *vista* (es decir, una consulta) sobre las ontologías locales. El significado de asociar a un término C en $\mathcal{G}$ una consulta $q_\mathcal{S}$ sobre $\mathcal{S}$, es que tal consulta representa la mejor manera de caracterizar a las instancias de C usando los conceptos en $\mathcal{S}$. Un mecanismo para caracterizar usado para especificar si la correspondencia entre C y la vista asociada es *sensata*, *completa* o *exacta*. Sea $\mathcal{D}$ un modelo local para $\mathcal{O}$ e $\mathcal{I}$ una interpretación global para $\mathcal{O}$:

- *sensata*: $\mathcal{I}$ satisface la correspondencia $\langle C, q_\mathcal{S}, sound\rangle$ en $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ con respecto a $\mathcal{D}$, si todas las tuplas que satisfacen $q_\mathcal{S}$ en $\mathcal{D}$ satisfacen C en $\mathcal{I}$, *i.e.*, $q_\mathcal{S}^\mathcal{D} \subseteq C^\mathcal{I}$;
- *completa*: $\mathcal{I}$ satisface la correspondencia $\langle C, q_\mathcal{S}, complete\rangle$ en $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ con respecto a $\mathcal{D}$, si no existe otra tupla que aquellas que satisfacen $q_\mathcal{S}$ en $\mathcal{D}$ satisface C en $\mathcal{I}$, *i.e.*, $q_\mathcal{S}^\mathcal{D} \supseteq C^\mathcal{I}$;
- *exacta*: $\mathcal{I}$ satisface la correspondencia $\langle C, q_\mathcal{S}, exact\rangle$ en $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ con respecto a $\mathcal{D}$, si el conjunto de tuplas que satisface C en $\mathcal{I}$ es exactamente el conjunto de tuplas que satisfacen $q_\mathcal{S}$ en $\mathcal{D}$, *i.e.* $q_\mathcal{S}^\mathcal{D} = C^\mathcal{I}$.

Se dice que $\mathcal{I}$ satisface la correspondencia $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ con respecto a $\mathcal{D}$, si $\mathcal{I}$ satisface cada correspondencia en $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ con respecto a $\mathcal{D}$.

Note también que las caracterizaciones lógicas de las vistas sensatas y completas en GAV son a través de las aserciones de primer orden $\forall x.q_\mathcal{S}(x) \rightarrow C(x)$ y $\forall x.C(x) \rightarrow q_\mathcal{S}(x)$, respectivamente. Esta consideración tomará importancia cuando reinterpretemos la integración de ontologías posiblemente inconsistentes en el contexto de la argumentación rebatible (véase la Sección 4.6).

2.5.3. Integración de abajo hacia arriba o *local-as-view*

En el acercamiento *local-as-view* (LAV) a la integración de ontologías, se asume que se tiene un lenguaje de consultas $\mathcal{V}_\mathcal{G}$ sobre el alfabeto $\mathcal{A}_\mathcal{G}$, y el mapeo entre la ontología global y las locales es dado al asociar a cada término de las ontologías locales una *vista*, *i.e.* una consulta, sobre la ontología global. El significado de la asociación entre un término C de $\mathcal{S}$ y de una consulta $q_\mathcal{G}$ sobre $\mathcal{G}$ es que tal consulta representa la mejor manera de caracterizar las instancias de C usando los conceptos en $\mathcal{G}$. De la misma manera que en el acercamiento *global-as-view*, la correspondencia entre C y la vista asociada puede ser sensata, completa o exacta. Sea $\mathcal{D}$ un modelo local para $\mathcal{O}$, e $\mathcal{I}$ una interpretación global para $\mathcal{O}$:

- *sensata*: $\mathcal{I}$ satisface la correspondencia $\langle q_{\mathcal{G}}, C, sound\rangle$ en $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ con respecto a $\mathcal{D}$, si todas las tuplas que satisfacen C en $\mathcal{D}$, satisfacen $q_{\mathcal{G}}$ en $\mathcal{I}$, *i.e.* $C^{\mathcal{I}} \subseteq q_{\mathcal{G}}^{\mathcal{D}}$;

- *completa*: $\mathcal{I}$ satisface la correspondencia $\langle q_{\mathcal{G}}, C, complete\rangle$ en $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ con respecto a $\mathcal{D}$, si ninguna tupla distinta a aquellas que satisfacen C en $\mathcal{D}$ satisfacen $q_{\mathcal{G}}$ en $\mathcal{I}$, *i.e.* $C^{\mathcal{I}} \supseteq q_{\mathcal{G}}^{\mathcal{D}}$;

- *exacta*: $\mathcal{I}$ satisface la correspondencia $\langle q_{\mathcal{G}}, C, exact\rangle$ en $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ con respecto a $\mathcal{D}$, si el conjunto de tuplas que satisfacen C en $\mathcal{D}$ es exactamente el conjunto de tuplas que satisfacen $q_{\mathcal{G}}$ en $\mathcal{I}$, *i.e.* $C^{\mathcal{I}} = q_{\mathcal{G}}^{\mathcal{D}}$.

Se dice que $\mathcal{I}$ satisface la correspondencia $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ con respecto a $\mathcal{D}$, si $\mathcal{I}$ satisface cada correspondencia en $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ con respecto a $\mathcal{D}$.

De la misma manera que en el acercamiento GAV, en el acercamiento LAV las caracterizaciones lógicas de las vistas sensatas y completas son a través de las aserciones de primer orden $\forall x.C(x) \rightarrow q_{\mathcal{G}}(x)$ y $\forall x.q_{\mathcal{G}}(x) \rightarrow C(x)$, respectivamente. Esta consideración también tomará importancia cuando reinterpretemos la integración de ontologías posiblemente inconsistentes en el contexto de la argumentación rebatible (véase la Sección 4.6).

El mayor desafío de este acercamiento es que para contestar una consulta expresada sobre el esquema global, uno debe ser capaz de reformular la consulta en términos de consultas a las fuentes. Mientras que en el acercamiento *global-as-view* tal reformulación es guiada por las correspondencias en el mapeo, aquí el problema requiere un *paso de razonamiento* para inferir cómo usar las fuentes para contestar la consulta. Muchos autores (Calvanese et al., 2001; Lenzerini, 2002) opinan que, a pesar de su dificultad, el acercamiento *local-as-view* soporta mejor un ambiente dinámico, donde las ontologías locales pueden ser agregadas a los sistemas sin necesidad de restructurar la ontología global. Este acercamiento ha sido aplicado a la integración de bases de datos heterogéneas almacenadas en almacenes de datos (en inglés, *data-warehouses*) (Alasoud et al., 2005).

2.6. Resumen

En este capítulo se presentaron los fundamentos de la Web Semántica y el razonamiento con ontologías. Las ontologías representan la descripción de una conceptualización de una parte de la realidad. Las ontologías jugarán un rol muy importante en la concepción de la Web Semántica ya que permiten definir con exactitud el significado de los datos en los recursos web. De esta manera, agentes con poder delegado por sus usuarios serán capaces de interpretar los datos allí presentes y razonar sobre el contenido informacional de los mismos. La descripción de las ontologías se realizará en la Web Semántica utilizando un lenguaje estándar llamado OWL que tiene su semántica basada en un formalismo lógico llamado Lógicas para la Descripción.

Un problema que surge al trabajar con ontologías es la presencia de incompletitud e inconsistencia de la información. Vimos las diversas situaciones que pueden hacer surgir la inconsistencia en la definición de ontologías. Se estudió un problema relacionado como ser la integración de ontologías, que se refiere a la combinación de dos o más ontologías en una única ontología, y que también puede hacer surgir inconsistencias ya que las distintas ontologías usualmente han sido desarrolladas por distintos autores con diferentes concepciones del dominio de aplicación.

Si bien existen razonadores capaces de interpretar definiciones ontológicas consistentes, los mismos sólo son capaces de detectar la presencia de inconsistencias y no son capaces de brindar mayores resultados. En el siguiente capítulo, exploraremos un formalismo alternativo para razonar con ontologías expresadas en Lógicas para la Descripción posiblemente inconsistentes.

Capítulo 3

Razonamiento no monótono con argumentación rebatible

La lógica matemática fue desarrollada para formalizar hechos precisos y razonamiento válido (o correcto). Cuando un agente razona utilizando lógica matemática pura, se asume que todos los hechos relevantes son conocidos a priori. Sin embargo, cuando lidiamos con el mundo real, esta hipótesis no es cierta ya que usualmente no contamos con todos los hechos de la realidad. Además, la lógica matemática es de naturaleza monótona; esto quiere decir que cuando agregamos nuevo conocimiento no es posible descartar conocimiento adquirido previamente.

En la matemática el conocimiento se representa con lógica de primer orden (FOL), el cual es un poderoso formalismo de representación de conocimiento. Sin embargo, la FOL tiene severas restricciones para las aplicaciones de Inteligencia Artificial (IA). Estas restricciones incluyen la monotonicidad y su limitada habilidad para representar información incierta e incompleta. Además, se presentan problemas cuando la información que se posee es contradictoria. Así, el razonamiento de sentido común es la habilidad de razonar en base a información incompleta y cambiar nuestras mentes cuando aparece nueva información (razonamiento no-monótono).

Las motivaciones para el estudio de los sistemas argumentativos pueden encontrarse en diversos trabajos de la literatura, como (Clark, 1990), (Carbogim et al., 2000), (Prakken and Vreeswijk, 2002), (Chesñevar, Maguitman and Loui, 2000) y (Bench-Capon and Dunne, 2007). La argumentación se preocupa de resolver los problemas anteriores. La argumentación surgió en el campo de la filosofía, y evolucionó desde un enfoque *informal* hacia distintos formalismos. Los sistemas formales de argumentación se caracterizan por representar los mismos elementos que los sistemas informales usando lenguajes formales y aplicando reglas de inferencias formales en ellos (Carbogim et al., 2000). Carbogim et al. (2000) (citando a Krause et al. (1995)) aclaran que:

"los argumentos tienen la forma de la demostración lógica pero no su fuerza"

queriendo decir que los argumentos tienen el mismo "sabor" que las demostraciones lógicas pero sus conclusiones no son necesariamente verdades en el sentido de la lógica clásica.

En este capítulo introduciremos a la argumentación rebatible. En particular, describiremos en detalle el formalismo de la Programación en Lógica Rebatible (DeLP). DeLP es un formalismo de razonamiento no monótono basado en la Programación en Lógica y en la argumentación rebatible. El lector debe notar que DeLP no es el único sistema de argumentación existente, otros sistemas han sido estudiados extensamente en las recopilaciones de Carbogim (Carbogim et al., 2000), Chesñevar *et al.* (Chesñevar, Maguitman and Loui, 2000) y Prakken y Vreeswijk (Prakken and Vreeswijk, 2002) en conjunción con otros trabajos particulares que se citarán oportunamente. También, muchos sistemas han sido estudiados en la Tesis de Magister (Gómez, 2003). En particular, consideramos a DeLP relevante ya que al ser un sistema con una implementación concreta y eficiente lo usaremos para ejemplificar muchas de las ideas que sustentan esta Tesis Doctoral.

El resto del capítulo está estructurado como sigue. En la Sección 3.1 se presentan los fundamentos del razonamiento no monótono, en la Sección 3.2 se presenta a la argumentación rebatible; y finalmente, en la Sección 3.3, se presenta a la Programación en Lógica Rebatible.

3.1. Razonamiento no monotónico

El *razonamiento monótono*, como aquel brindado por la Lógica de Primer Orden, asume un mundo incremental, donde los nuevos estados y situaciones son agregados a los estados corrientes y situaciones. Formalmente, un sistema es *monótono* (o *monotónico*) si, a partir de un conjunto de premisas con una conclusión, la misma conclusión puede ser derivada a partir de cada superconjunto del conjunto original de premisas. Sin embargo, el mundo real no opera de esta manera.

Para crear sistemas inteligentes que puedan razonar acerca del mundo real se han propuesto una variedad de estilos de razonamiento (Brewka et al., 1997). La idea consiste en describir el razonamiento de sentido común, en éste se asume que el mundo es dinámico, con nuevos estados de conocimiento surgiendo a partir de los viejos. Como tales estados reemplazan a los viejos, y los nuevos estados deben ser representados de una manera tal que los viejos estados sean cancelados.

Así, el *razonamiento no monótono* (o *no monotónico*) es el razonamiento en la base de información incompleta (Brewka et al., 1997). Dada más información, estamos preparados para retractar inferencias realizadas previamente. Para exhibir el ejemplo clásico: si todos sabemos que Tweety es un ave, entonces podemos concluir plausiblemente que puede volar; al aprender que Tweety es un pingüino, descartaremos tal conclusión. Llamamos a este tipo de razonamiento no monotónico porque el conjunto de conclusiones plausibles no crece monotónicamente al crecer la información disponible.

En particular, en esta Tesis utilizaremos la argumentación rebatible como sistema principal de razonamiento no monótono. Sin embargo, introduciremos a continuación dos nociones que serán de utilidad en el resto de la presentación.

Definición 3.1.1 (Hipótesis de mundo cerrado (Brewka et al., 1997)) *Dada una base de conocimiento, la hipótesis de mundo cerrado (o CWA por Closed World Assumption) intuitivamente significa que cualquier información no mencionada en la base de conocimiento es considerada como falsa. Más precisamente, si una instancia positiva de un predicado no es contenida en la base de datos, su negación es asumida como verdadera.*

Definición 3.1.2 (Hipótesis de nombres únicos (Brewka et al., 1997)) *Dada una base de conocimiento, la hipótesis de nombres únicos (o UNA por Unique Name Assumption) asume que individuos distintos tienen nombres distintos.*

Nótese que en la Web Semántica, al usar URLs y URIs para identificar individuos y clases, se cumple la hipótesis de nombres únicos.

Definición 3.1.3 (Hipótesis de dominio cerrado (Brewka et al., 1997)) *Dada una base de conocimiento, la hipótesis de dominio cerrado (o DCA por Domain Closed Assumption) quiere decir que el número de individuos de tal base de conocimiento es finito.*

3.2. Argumentación rebatible

Uno de los desafíos de la Inteligencia Artificial es modelar el razonamiento de sentido común, el cual casi siempre ocurre en presencia de información incompleta y potencialmente inconsistente (McCarthy and Hayes, 1969; Reiter, 1980; McCarthy, 1990). Los modelos lógicos del razonamiento de sentido común demandan la formalización de principios y criterios para caracterizar patrones válidos de inferencia. En este respecto, la lógica clásica ha probado ser inadecuada, ya que se comporta *monotónicamente* y no permite lidiar con inconsistencias a nivel objeto (Reiter, 1980).

Cuando una regla apoyando una conclusión puede ser derrotada por nueva información, se dice que tal razonamiento es *rebatible* (Pollock, 1974, 1987; Nute, 1988; Pollock, 1995; Simari and Loui, 1992). Cuando tales razones rebatibles o reglas son encadenadas para llegar a una conclusión, tenemos *argumentos* en lugar de pruebas. Los argumentos pueden competir, derrotándose entre sí, de tal manera que un *proceso* de argumentación es el resultado natural de la búsqueda de argumentos. La adjudicación de argumentos que compiten entre sí debe ser realizada, comparando argumentos para determinar qué creencias están últimamente aceptadas como *garantizadas* o *justificadas*. La preferencia entre argumentos en conflicto es definida en términos de un *criterio de preferencia* que establece una relación " $\preceq$ " entre argumentos posibles; así, para dos argumentos $\mathcal{A}$ y $\mathcal{B}$ en

conflicto, puede ser el caso que $\mathcal{A}$ es estrictamente preferido sobre $\mathcal{B}$ ($\mathcal{A} \succ \mathcal{B}$), que $\mathcal{A}$ y $\mathcal{B}$ son igualmente preferidos ($\mathcal{A} \succeq \mathcal{B}$ y $\mathcal{A} \preceq \mathcal{B}$) o que $\mathcal{A}$ y $\mathcal{B}$ no son comparables entre sí. En el marco descripto, como arribamos a conclusiones por medio de la construcción de argumentos rebatibles, y como la *argumentación lógica* es usualmente referida como *argumentación*, llamaremos a este tipo de razonamiento *argumentación rebatible*.

Consideremos un problema bien conocido de razonamiento no monótono en IA acerca de la habilidades voladoras de aves, expresado en términos argumentativos. Consideremos las siguientes sentencias:

1. Las aves usualmente vuelan.
2. Los pingüinos usualmente no vuelan.
3. Los pingüinos son aves.

Las dos primeras sentencias corresponden a *reglas rebatibles* (reglas que están sujetas a posibles excepciones). La tercera sentencia es una *regla estricta*, donde no hay excepciones posibles. Ahora, dado el hecho de que *Tweety es un pingüino*, dos argumentos diferentes pueden ser construidos:

1. El argumento $\mathcal{A}$ (basado en las reglas 1 y 3): Tweety es un pingüino. Los pingüinos son aves. Las aves usualmente vuelan. Luego, Tweety vuela.
2. El argumento $\mathcal{B}$ (basado en la regla 2): Tweety es un pingüino. Los pingüinos usualmente no vuelan. Luego, Tweety no vuela.

En esta situación particular, los dos argumentos que surgen no pueden ser aceptados simultáneamente (ya que arriban a conclusiones contradictorias). Nótese que el argumento $\mathcal{B}$ parece racionalmente preferible al argumento $\mathcal{A}$, ya que está basado en información más *específica*. De hecho, la especificidad es comúnmente adoptada como un criterio de preferencia sintáctico entre argumentos en conflicto prefiriendo aquellos argumentos que son *más informados* o *más directos* (Poole, 1985, 1989). En este caso particular, si adoptamos la especificidad como criterio de preferencia, el argumento $\mathcal{B}$ está justificado, mientras que $\mathcal{A}$ no lo está (ya que es derrotado por $\mathcal{B}$). La situación descripta puede fácilmente volverse mucho más compleja, como un argumento puede ser derrotado por un tercer argumento, *reinstituyendo* al primero; por ejemplo, podríamos enterarnos que Tweety es en realidad un pingüino genéticamente alterado y podríamos saber también que los pingüinos genéticamente alterados son capaces de volar).

El éxito creciente de los acercamientos basados en argumentación ha causado un rico cruzamiento con otras disciplinas, proveyendo interesantes resultados en diferentes áreas como toma de decisiones en grupo (Zhang et al., 2005), ingeniería de conocimiento (Carbogim et al., 2000), razonamieno legal (Prakken and Sartor, 2002; Verheij, 2005) y sistemas

multiagente (Parsons et al., 1998; Sierra and Noriega, 2002; Rahwan et al., 2003), entre otros (Chesñevar, Maguitman and Loui, 2000). Durante la última década, varios marcos de argumentación rebatible han sido desarrollados, muchos de ellos sobre la base de apropiadas extensiones a la programación en lógica (ver (Chesñevar, Maguitman and Loui, 2000; Prakken and Vreeswijk, 2002; Kakas and Toni, 1999)). La *Programación en Lógica Rebatible* (DeLP) (García and Simari, 2004) es uno de tales formalismos, combinando resultados de la teoría de la argumentación rebatible (Simari and Loui, 1992) y la programación en lógica. DeLP es un marco apropiado para construir aplicaciones del mundo real que han probado ser particularmente atractivas en tal contexto, tales como agrupamiento de patrones (Gómez and Chesñevar, 2004), gestión de conocimiento (Chesñevar, Brena and Aguirre, 2005), sistemas multiagente (Brena et al., 2006), entre otros.

3.2.1. Generalidades

La argumentación provee una perspectiva diferente al razonamiento no-monótono y al razonamiento rebatible, en el cual una afirmación es aceptada o rechazada en la base de los argumentos en favor o en contra de ella, y en el hecho de si dichos argumentos son atacados y derrotados por otros (Carbogim et al., 2000). Esta visión es sustentada por la *argumentación rebatible*.

Los marcos (*frameworks* en inglés) de argumentación tienen generalmente los siguientes elementos (Prakken and Vreeswijk, 2002; Chesñevar, Maguitman and Loui, 2000): un lenguaje lógico subyacente, un concepto de argumento, un concepto de conflicto entre argumentos, una noción de derrota entre argumentos, y, una noción de cuándo un argumento es aceptable. A continuación, describiremos someramente a cada uno de los elementos de un sistema argumentativo.

Lenguaje lógico subyacente: El lenguaje lógico subyacente será el de la lógica formal de primer orden donde están definida una o más relaciones de consecuencia monótonas que establecen la base para derivar argumentos. En general, muchos sistemas de argumentación involucran una base de conocimiento $K = (\Pi, \Delta)$ que provee el conocimiento del dominio para un agente formalizado en un lenguaje de primer orden L. Este conocimiento de dominio involucra usualmente un conjunto Π de reglas estrictas y hechos y un conjunto Δ de reglas rebatibles.

Argumento: Un argumento es una prueba rebatible obtenida a partir de la base de conocimiento K por medio de la aplicación de convenientes reglas de inferencia (quizá rebatibles) asociadas con el lenguaje lógico subyacente L.

Conflicto entre argumentos: Dados dos argumentos $\mathcal{A}$ y $\mathcal{B}$, el *conflicto* (o *ataque*) entre argumentos surge siempre que $\mathcal{A}$ y $\mathcal{B}$ no puedan ser aceptados simultáneamente (típicamente debido a algún tipo de contradicción lógica).

Derrota entre argumentos: Como el lenguaje lógico subyacente es monótono, el agregado de nueva información no invalida argumentos existentes ni conclusiones derivadas previamente; por lo tanto, en una base de conocimiento pueden coexistir argumentos en conflicto. El carácter no-monótono de la argumentación surge del hecho que algunos argumentos serán preferidos sobre otros y el formalismo debería tener medios para decidir cuál de dichos argumentos es aceptable.

La noción de derrota está usualmente basada en alguna medida comparativa para los argumentos y se utiliza algún criterio basado en esta medida para decidir la identidad del argumento ganador. Una manera es asignar prioridad a las reglas en un sistema basado en reglas, otra forma son los criterios de especificidad y el de directitud.

Formalmente, muchos sistemas de argumentación proveen un criterio de preferencia que define un orden parcial entre argumentos, permitiendo determinar cuando $\mathcal{A}$ debe preferirse sobre $\mathcal{B}$. Esto define una relación de *derrota* (*defeats*). Dado el conjunto *Args* de argumentos obtenidos de una base de conocimiento K, vale que la relación de ataque $attacks \subseteq Args \times Args$. Cuando el argumento $\mathcal{A}$ se prefiere sobre un argumento $\mathcal{B}$, se dice que $\mathcal{A}$ derrota a $\mathcal{B}$. Además, vale que $defeats \subseteq attacks$.

Argumento aceptable: El objetivo de un sistema argumentativo es el de determinar qué afirmaciones y qué argumentos son aceptables. La noción de aceptabilidad varía de formalismo en formalismo, pero intuitivamente la noción se corresponde con que un argumento no será aceptable si es derrotado por algún argumento aunque sea capaz de derrotar a un argumento en conflicto con él. Para determinar si un argumento dado $\mathcal{A}$ es *aceptable* (o *triunfador* o *justificado*), se lleva a cabo un proceso dialéctico, en el cual se tienen en cuenta derrotadores para $\mathcal{A}$, derrotadores para estos derrotadores, y así sucesivamente. Por ello, para este proceso se deben tener en cuenta a todos los argumentos a favor y en contra de una afirmación dada antes de tomar una decisión.

La investigación que da marco a esta Tesis ha sido desarrollada sin ningún sistema de argumentación particular en mente. Sin embargo, debido a que la Programación en Lógica Rebatible es un sistema particular de la argumentación rebatible, el cual se halla implementado en forma eficiente, ejemplificaremos los resultados de tal investigación en dicho formalismo. Por ello, el resto de este capítulo se centra en una descripción detallada de ella. Para una descripción más profunda de otros formalismos de argumentación, el lector puede leer el resumen realizado en la Tesis de Magíster realizada previamente (Gómez, 2003).

3.3. Programación en Lógica Rebatible (DeLP)

En el capítulo 4 mostraremos cómo se pueden resolver la incompletitud e inconsistencia de la información expresada en una ontología o base de conocimiento expresada en un lenguaje de representación de ontologías mediante un método de razonamiento no monótono basado en argumentación rebatible. En particular, existen implementaciones eficientes de razonadores basados en este tipo de argumentación realizada en términos de la Programación en Lógica Rebatible.

Por ello, en esta sección, definiremos el formalismo de la *Programación en Lógica Rebatible* (DeLP por su denominación en inglés *Defeasible Logic Programming*). basado en el texto de García and Simari (2004). Definiremos primero el lenguaje del mismo y luego las reglas de razonamiento rebatible que lo caracterizan. Cuando sea necesario ejemplificaremos el comportamiento del sistema en aquel aspecto que se estuviera describiendo. Nótese que las definiciones dadas a continuación pueden encontrarse en los siguientes artículos (Simari, 1989; Simari and Loui, 1992; García, 1997; Chesñevar, Maguitman and Loui, 2000; García and Simari, 2004). Otros trabajos como (Chesñevar, Simari and García, 2000) enriquecen las ideas presentadas en esta sección.

3.3.1. El lenguaje

El lenguaje de la DeLP se define en términos de tres conjuntos disjuntos: un conjunto de *hechos*, un conjunto de *reglas estrictas* y un conjunto de *reglas rebatibles*. En el lenguaje de DeLP un literal "L" es un átomo fijo[1] "A" o un átomo fijo negado "$\sim A$", donde "$\sim$" representa la *negación fuerte*. Por lo tanto, los literales no tienen variables.

Definición 3.3.1 (Hecho) *Un hecho es un literal,* i.e. *un átomo fijo o un átomo fijo negado.*

Los hechos pueden ser relacionales (versando sobre individuos) o proposicionales (con aridad nula). Los hechos son fijos (no contienen variables) y usualmente son denotados con letras minúsculas a la usanza del lenguaje de programación Prolog. A continuación, ejemplificamos estos conceptos.

Ejemplo 3.3.1 *Los siguientes son dos hechos positivos que expresan que Tweety es un ave, que Opus es un pingüino, CBD05 es un avión y que Opus posee a CBD05.*

$$bird(tweety)$$
$$penguin(opus)$$
$$plane(cbd05)$$
$$owns(opus, cbd05)$$

[1] Fijo en inglés es *ground*.

El siguiente es un hecho negativo que expresa que Opus no es capaz de volar:

$$\sim \mathit{flies}(\mathit{opus})$$

Ahora mostramos un hecho proposicional que indica que una situación es extraña:

$$\mathit{strange}$$

Definición 3.3.2 (Regla estricta) *Una regla estricta es un par ordenado, denotado "Cabeza ← Cuerpo", cuyo primer miembro, Cabeza, es un literal y cuyo segundo miembro, Cuerpo, es un conjunto finito no vacío de literales. Una regla estricta con la cabeza L_0 y cuerpo $\{L_1, \ldots, L_n\}$ se puede escribir también como $L_0 \leftarrow L_1, \ldots, L_n (n > 0)$.*

Veamos ejemplos de reglas estrictas:

Ejemplo 3.3.2 *La siguiente regla expresa que los pingüinos son aves:*

$$\mathit{bird}(X) \leftarrow \mathit{penguin}(X)$$

Definición 3.3.3 (Regla rebatible) *Una regla rebatible es un par ordenado, denotado "Cabeza ≺ Cuerpo", cuyo primer miembro, Cabeza, es un literal y cuyo segundo miembro, Cuerpo, es un conjunto finito no vacío de literales. Una regla estricta con la cabeza L_0 y cuerpo $\{L_1, \ldots, L_n\}$ se puede escribir también como $L_0 \prec L_1, \ldots, L_n (n > 0)$.*

Sintácticamente, el símbolo " ≺ " es lo único que diferencia a una regla rebatible de una estricta. Pragmáticamente, una regla rebatible se usa para representar conocimiento rebatible, es decir, información tentativa que se puede usar si no hay nada en contra de ella. Así, mientras una regla estricta se usa para representar información no rebatible como: "*ave ← avestruz*" que expresa que "*todos los avestruces son aves*", una regla rebatible se usa para representar conocimiento como "*vuela ≺ ave*" que expresa que: "*las razones para creer que es un ave proveen razones para creer que puede volar*", o "*usualmente, las aves pueden volar*" (adaptado de (García and Simari, 2004)).

Veamos ejemplos de reglas rebatibles:

Ejemplo 3.3.3 *Las siguientes reglas expresan que usualmente las aves vuelan, que los pingüinos usualmente no vuelan y que un pingüino sea dueño de un avión es usualmente una situación extraña:*

$$\mathit{flies}(X) \prec \mathit{bird}(X)$$
$$\sim \mathit{flies}(X) \prec \mathit{penguin}(X)$$
$$\mathit{strange} \prec \mathit{penguin}(X), \mathit{owns}(X, Y), \mathit{plane}(Y)$$

Definición 3.3.4 (Programa lógico rebatible) *Un programa lógico rebatible $\mathcal{P}$, abreviado de.l.p., es un conjunto (posiblemente infinito) de hechos, reglas estrictas y reglas rebatibles. En un programa $\mathcal{P}$, distinguiremos el subconjunto Π de los hechos y reglas estrictas del subconjunto Δ de reglas rebatibles. Cuando se requiera, denotaremos $\mathcal{P}$ como (Π, Δ).*

Tanto las reglas estrictas como las rebatibles son fijas. Sin embargo, en la literatura, se estila usar "reglas esquemáticas" con variables. Como se nota habitualmente en la Programación en Lógica, los nombres de predicados se denotan con minúsculas, las variables se denotarán comenzando con letras mayúsculas y las constantes comenzando con letras minúsculas. En lo que sigue, los programas particulares en lugar de especificarse como pares ordenados, simplemente se expresarán como una secuencia de esquemas de reglas o metareglas como las descriptas anteriormente. A continuación, mostramos un ejemplo de programa lógico rebatible.

Ejemplo 3.3.4 *Sea $\mathcal{P}_{3.3.4} = (\Pi, \Delta)$ el siguiente programa DeLP tal que:*

$$\Pi = \left\{ \begin{array}{l} bird(tweety); \\ penguin(opus); \\ plane(cbd05); \\ owns(opus, cbd05); \\ bird(X) \leftarrow penguin(X) \end{array} \right\}, \text{ y}$$

$$\Delta = \left\{ \begin{array}{l} flies(X) \prec bird(X); \\ \sim flies(X) \prec penguin(X); \\ flies(X) \prec penguin(X), owns(X, Y), plane(Y); \\ strange \prec penguin(X), owns(X, Y), plane(Y) \end{array} \right\}.$$

El conjunto Π denotando la información estricta dice que Tweety es un ave ($bird(tweety)$), Opus es un pingüino ($penguin(opus)$), CBD05 es un avión ($plane(cbd05)$), Opus posee a CBD05 ($owns(opus, cbd05)$) y que los pingüinos son aves ($bird(X) \leftarrow penguin(X)$). Por otro lado, el conjunto Δ que representa la información tentativa o rebatible denota que las aves usualmente vuelan ($flies(X) \prec bird(X)$), si un individuo es un pingüino hay razones tentativas para creer que es incapaz de volar ($\sim flies(X) \prec penguin(X)$) a menos que posea un avión ($flies(X) \prec penguin(X), owns(X, Y), plane(Y)$), y ésta es una situación extraña ($strange \prec penguin(X), owns(X, Y), plane(Y)$).

Definición 3.3.5 (Derivación rebatible) *Sea $\mathcal{P} = (\Pi, \Delta)$ un programa DeLP y L un literal fijo. Una derivación rebatible de L a partir de $\mathcal{P}$, denotada $\mathcal{P} \vdash L$, consiste de una secuencia finita $L_1, L_2, \ldots, L_n = L$ de literales, y cada literal L_i está en la secuencia porque:*

- *L_i es un hecho en Π, o,*

- *existe una regla R_i en $\mathcal{P}$ (estricta o rebatible) con cabeza L_i y cuerpo $B_1, B_2, \ldots, B_k$ y cada literal del cuerpo es un elemento L_j de la secuencia apareciendo antes de L_i tal que $(j < i)$.*

Definición 3.3.6 (Derivación estricta) *Sea $\mathcal{P}$ un programa DeLP y H un literal con una derivación rebatible $L_1, L_2, \ldots, L_n = h$. Se dice que H tiene una derivación estricta a partir de $\mathcal{P}$, denotada $\mathcal{P} \vdash L$, si o bien H es un hecho o todas las reglas usadas para obtener la secuencia $L_1, L_2, \ldots, L_n$ son reglas estrictas.*

Formalmente y de acuerdo a la Definición 3.3.5, las derivaciones son consideradas como listas o secuencias. Además, debido al parentesco de DeLP con la programación en lógica tradicional, las derivaciones se pueden representar como un árbol SLD (Sterling and Shapiro, 1994). Sin embargo, se las suele representar como árboles donde la raíz consiste de la cabeza de una regla y sus hijos de los antecedentes de la regla y así sucesivamente, como muestran Chesñevar, Maguitman and Loui (2000, Fig.˜2, pág.˜355).

Ejemplo 3.3.5 *Consideremos nuevamente el programa DeLP $\mathcal{P}_{3.3.4} = (\mathcal{P}, \Delta)$ presentado en el Ejemplo 3.3.4 de la página 69. Vemos que el literal flies(tweety) se deduce a partir de $\mathcal{P}_{3.3.4}$ a través de la siguiente secuencia de literales: [bird(tweety), flies(tweety)]. Además, el literal flies(opus) se deduce también a partir de $\mathcal{P}_{3.3.4}$ a través de la secuencia de literales [penguin(opus), bird(opus), flies(opus)]. Sin embargo, esta situación queda más claramente representada por el árbol de la Figura 3.1. Vemos también que no existe una única derivación para el literal flies(opus), éste también puede ser derivado por: [penguin(opus), owns(opus, cbd05), plane(cbd05), flies(opus)]. Vemos la derivación en forma gráfica en la Figura 3.2 de la página 71 .*

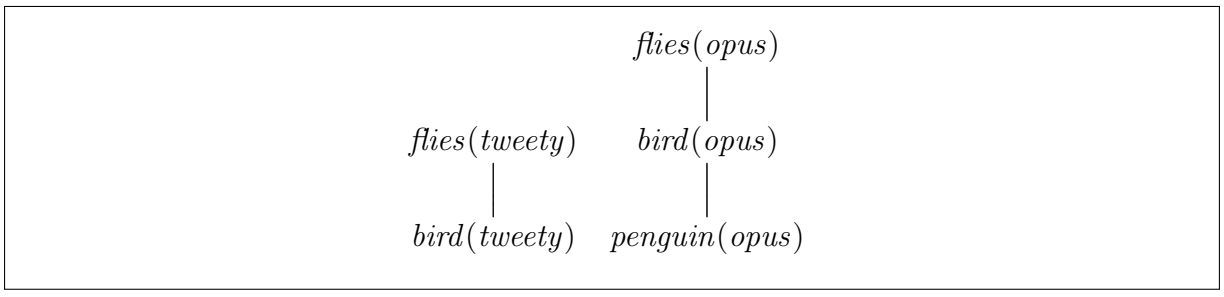


Figura 3.1: Derivación de literales en DeLP

El símbolo "˜" se utiliza para denotar el complemento de un literal con respecto a la negación fuerte, es decir $\overline{P}$ es $\sim P$ y $\overline{\sim P}$ es P. Dos literales son contradictorios si son complementarios.

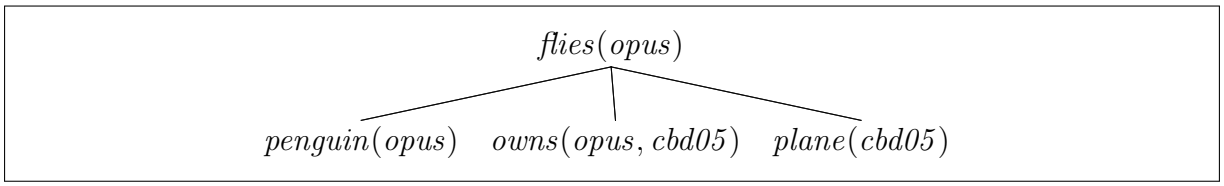


Figura 3.2: Derivación de literales en DeLP

Definición 3.3.7 (Conjunto contradictorio de reglas) *Un conjunto de reglas es contradictorio si y sólo si existe una derivación rebatible para un par de literales complementarios a partir de este conjunto.*

Veamos un ejemplo de este concepto.

Ejemplo 3.3.6 *Consideremos nuevamente el programa DeLP $\mathcal{P}_{3.3.4} = (\Pi, \Delta)$ presentado en el Ejemplo 3.3.4 de la página 69. El conjunto $\Pi \cup \Delta$ es un conjunto contradictorio de reglas ya que permite la derivación de los literales complementarios flies(opus) y ∼flies(opus). Dicha situación puede apreciarse en la Figura 3.3 de la página 71.*

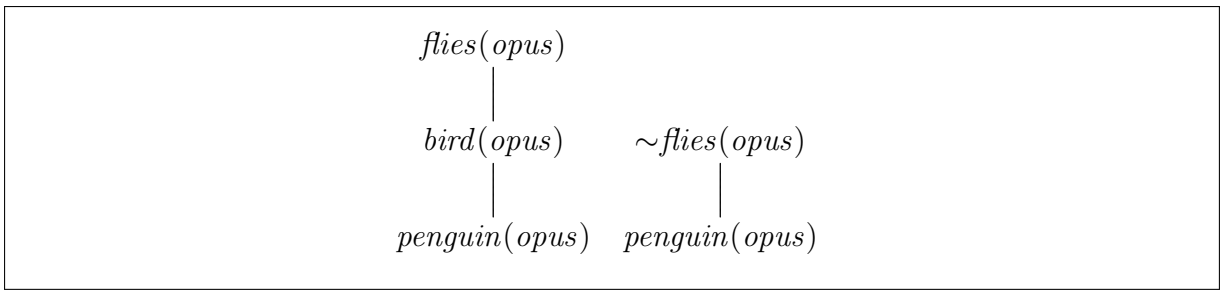


Figura 3.3: Derivación de literales complementarios en DeLP

3.3.2. Argumentación rebatible

El marco (*framework*) de la Programación en Lógica Rebatible se basa en la noción de argumento. Un argumento es básicamente un conjunto de razones, no contradictorio y minimal con respecto a la inclusión de conjuntos, a favor de una conclusión.

Definición 3.3.8 (Estructura de argumento) *Sea H un literal y $\mathcal{P} = (\Pi, \Delta)$ un programa. Se dice que $\langle \mathcal{A}, H \rangle$ es una estructura de argumento para H, si $\mathcal{A}$ es un conjunto de reglas rebatibles de Δ tal que:*

1. *existe una derivación rebatible para H a partir de $\Pi \cup \mathcal{A}$, notado como $\Pi \cup \mathcal{A} \sim h$;*
2. *el conjunto $\Pi \cup \mathcal{A}$ no es contradictorio, notado como $\Pi \cup \mathcal{A} \not\sim \bot$, y,*

3. *$\mathcal{A}$ es minimal: no hay ningún subconjunto propio $\mathcal{A}'$ de $\mathcal{A}$ tal que $\mathcal{A}'$ satisface las condiciones (1) y (2).*

Una estructura de argumento $\langle \mathcal{A}, H \rangle$, o simplemente, un *argumento* $\mathcal{A}$ para H, es un conjunto minimal no contradictorio de reglas rebatibles, obtenido a partir de una derivación rebatible para un dado literal H. El literal H también se llamará la 'conclusión' soportada por $\mathcal{A}$.

Veamos el siguiente ejemplo para estudiar la noción de argumento.

Ejemplo 3.3.7 *Consideremos nuevamente el programa DeLP $\mathcal{P}_{3.3.4} = (\Pi, \Delta)$ presentado en el Ejemplo 3.3.4 de la página 69. En dicho programa, tenemos un argumento $\langle \mathcal{A}_1, flies(tweety) \rangle$ a favor de que Tweety vuela basado en las premisas que el mismo es un ave, donde:*

$$\mathcal{A}_1 \;=\; \left\{\; flies(tweety) \prec bird(tweety) \;\right\}.$$

Además, tenemos tres argumentos $\langle \mathcal{B}_1, flies(opus) \rangle$, $\langle \mathcal{B}_2, \sim flies(opus) \rangle$ y $\langle \mathcal{B}_3, flies(opus) \rangle$:

$$\begin{array}{rcl} \mathcal{B}_1 & = & \left\{\; flies(opus) \prec bird(opus) \;\right\}; \\ \mathcal{B}_2 & = & \left\{\; \sim flies(opus) \prec penguin(opus) \;\right\}, \text{ y} \\ \mathcal{B}_3 & = & \left\{\; flies(opus) \prec penguin(opus), owns(opus, cbd05), plane(cbd05) \;\right\}. \end{array}$$

El argumento $\mathcal{B}_1$ expresa que Opus debe volar porque es un ave, mientras que el argumento $\mathcal{B}_2$ dice que Opus no vuela porque es un pingüino. Por otro lado, el argumento $\mathcal{B}_3$ soporta la conclusión que Opus vuela ya que es dueño de un avión (el CBD05).

Otro argumento que se puede construir a partir de este programa es $\langle \mathcal{C}, strange \rangle$, donde:

$$\mathcal{C} \;=\; \left\{\; strange \prec penguin(opus), owns(opus, cbd05), plane(cbd05) \;\right\},$$

que expresa que es una situación extraña que un pingüino sea dueño de un avión.

Nótese como las reglas estrictas (y, por lo tanto, los hechos) no se incluyen en el conjunto de soporte de los argumentos. Es importante notar que un argumento $\langle \mathcal{A}, H \rangle$ se representa esquemáticamente como un triángulo con base $\mathcal{A}$ (el conjunto de soporte) y vértice H (el literal concluido). La intuición detrás de esta notación es la de resumir el árbol de derivación del argumento como el de las Figuras 3.1, 3.2 y 3.3. Cuando querramos graficar un argumento sin referirnos a su estructura interna, lo haremos como se aprecia en la Figura 3.4 de la página 73.

Note que los argumentos no son cerrados bajo unión, ya que al unir argumentos podemos obtener conjuntos de reglas contradictorios o no minimales.

Definición 3.3.9 (Subargumento) *Sean $\langle \mathcal{A}, H \rangle$ y $\langle \mathcal{B}, Q \rangle$ dos estructuras de argumentos. $\langle \mathcal{B}, Q \rangle$ es un subargumento de $\langle \mathcal{A}, H \rangle$ si $\mathcal{B} \subseteq \mathcal{A}$.*

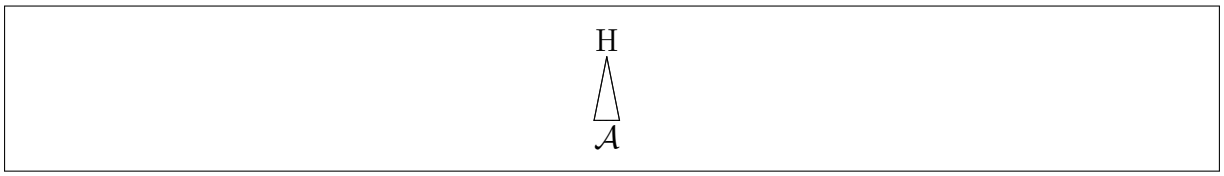


Figura 3.4: Argumento $\langle \mathcal{A}, H \rangle$ representado esquemáticamente

Consideremos el siguiente ejemplo sobre la definición recién dada.

Ejemplo 3.3.8 *Sea* $\mathcal{P}_{3.3.8} = (\Pi, \Delta)$ *el siguiente programa DeLP:*

$$\Pi = \left\{ \begin{array}{l} seats\_in\_computer(john) \\ seats\_in\_computer(paul) \\ uses\_browser(paul) \\ uses\_browser(X) \leftarrow checks\_web\_mail(X) \\ seats\_in\_computer(mary) \\ checks\_web\_mail(mary) \\ reads\_javadoc(mary) \end{array} \right\}, y$$

$$\Delta = \left\{ \begin{array}{l} pass(X) \prec studies(X) \\ studies(X) \prec seats\_in\_computer(X) \\ \sim studies(X) \prec seats\_in\_computer(X), web\_surfing(X) \\ \sim pass(X) \prec \sim studies(X) \\ web\_surfing(X) \prec uses\_browser(X) \\ \sim web\_surfing(X) \prec uses\_browser(X), reads\_javadoc(X) \end{array} \right\}.$$

El conjunto de información estricta Π *expresa que John ha sido visto sentado en su computadora, lo mismo ha ocurrido con Paul. Además, Paul ha sido visto usando un web browser. Todos los usuarios que revisan el correo electrónico via web usan un web browser. También, Mary ha sido vista sentada en la computadora, revisando el correo electrónico via web y estudiando documentación Javadoc .*

El conjunto de información tentativa Δ *establece que los alumnos que estudian usualmente aprueban el examen; los alumnos que han sido vistos sentados en la computadora usualmente están estudiando a menos que estén navegando; los alumnos que no estudian generalmente no aprueban el examen y los alumnos que usan un web browser usualmente están navegando a menos que lean documentación Javadoc.*

A partir del programa DeLP $\mathcal{P}_{3.3.8}$ *podemos obtener el argumento* $\langle \mathcal{A}_1, pass(john) \rangle$ *a favor de que John pasará el examen, donde:*

$$\mathcal{A}_1 = \left\{ \begin{array}{l} pass(john) \prec studies(john); \\ studies(john) \prec seats\_in\_computer(john) \end{array} \right\}.$$

Si consideramos el argumento $\langle \mathcal{B}_1, studies(john) \rangle$*, donde:*

$$\mathcal{B}_1 = \left\{ \; studies(john) \prec seats\_in\_computer(john) \; \right\},$$

vemos que $\langle \mathcal{B}_1, studies(john)\rangle$ *es un subargumento de* $\langle \mathcal{A}_1, pass(john)\rangle$. *Gráficamente, la situación se muestra en la Figura 3.5 de la página 74. Nótese como un subargumento corresponde a una porción de la derivación de un argumento. En la parte (a) de la figura se muestra la derivación de literal* $pass(john)$*; en la parte (b) se muestra qué parte de la derivación en (a) corresponde al argumento* $\mathcal{A}_1$*, y en la parte (c) se muestra qué parte de la misma derivación corresponde al subargumento* $\mathcal{B}_1$.

De la misma manera, tenemos un argumento $\langle \mathcal{A}_2, \sim pass(paul)\rangle$ *a favor de la conclusión tentativa que dice que Paul no aprobará el examen, donde:*

$$\mathcal{A}_2 = \left\{ \begin{array}{l} \sim pass(paul) \prec \sim studies(paul); \\ \sim studies(paul) \prec seats\_in\_computer(paul), web\_surfing(paul); \\ web\_surfing(paul) \prec uses\_browser(paul) \end{array} \right\}.$$

Claramente, el argumento $\langle \mathcal{B}_2, \sim studies(paul)\rangle$ *es un subargumento de* $\langle \mathcal{A}_2, \sim pass(paul)\rangle$ *donde* $\mathcal{A}_2$ *está formado por los dos últimos elementos del conjunto* $\mathcal{A}_2$ *mostrado arriba:*

$$\mathcal{B}_2 = \left\{ \begin{array}{l} \sim studies(paul) \prec seats\_in\_computer(paul), web\_surfing(paul); \\ web\_surfing(paul) \prec uses\_browser(paul) \end{array} \right\}.$$

La situación puede ser apreciada gráficamente en la Figura 3.6 de la página 75. De nuevo, en (a) se muestra la derivación de $\sim pass(paul)$*; en (b), el esquema del argumento* $\mathcal{A}_2$*, y, en (c), el esquema del subargumento* $\mathcal{B}_2$.

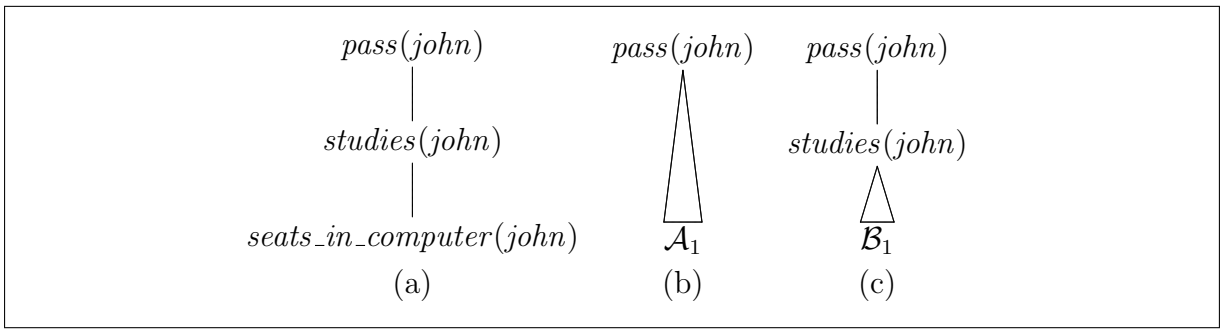


Figura 3.5: Subargumento $\langle \mathcal{B}_1, studies(john)\rangle$ de $\langle \mathcal{A}_1, pass(john)\rangle$ respecto del programa DeLP $\mathcal{P}_2 = (\Pi_2, \Delta_2)$

3.3.3. Contraargumentos

En la DeLP, las respuestas a las consultas están soportadas por argumentos. Sin embargo, estos argumentos pueden ser derrotados por otros argumentos. Informalmente, una consulta Q tendrá éxito si el argumento de soporte para ella no está derrotado. Para establecer si $\mathcal{A}$ no es un argumento derrotado, se deben considerar los contraargumentos de $\mathcal{A}$ que podrían convertirse en *derrotadores* de $\mathcal{A}$. En síntesis, un argumento $\mathcal{B}$, que

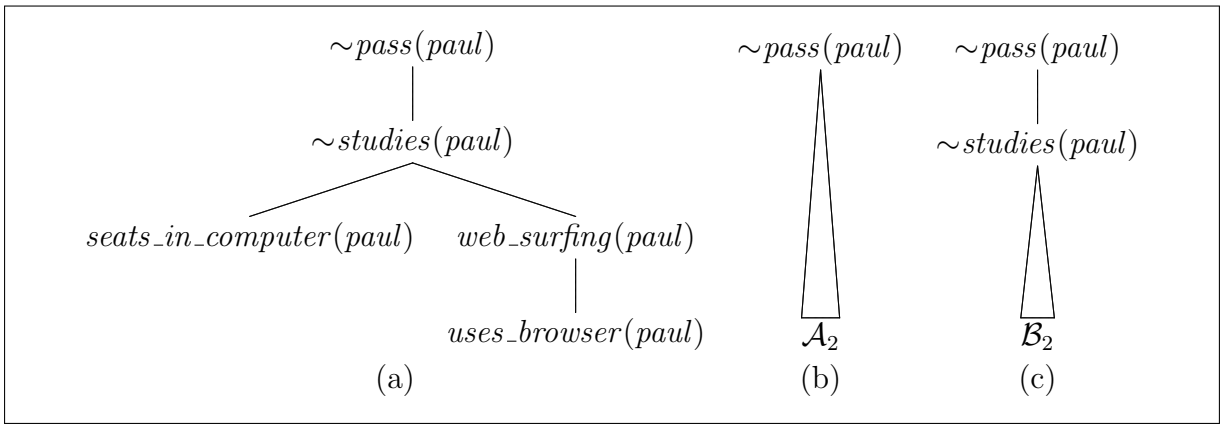


Figura 3.6: Subargumento $\langle \mathcal{B}_2, \sim studies(paul)\rangle$ de $\langle \mathcal{A}_2, \sim pass(paul)\rangle$ respecto del programa DeLP $\mathcal{P}_2 = (\Pi_2, \Delta_2)$

sea a su vez contraargumento de otro argumento $\mathcal{A}$, se convertirá en un derrotador de $\mathcal{A}$ si lo contradice y además es mejor. A continuación, definiremos qué quiere decir que un argumento contradice a otro y qué quiere decir que un argumento es mejor que otro.

Definición 3.3.10 (Desacuerdo) *Sea $\mathcal{P} = \Pi \cup \Delta$ un programa DeLP. Se dice que dos literales H_1 y H_2 están en desacuerdo si y sólo si el conjunto $\Pi \cup \{H_1, H_2\}$ es contradictorio.*

Dos literales complementarios están trivialmente en desacuerdo. Sin embargo, dos literales no complementarios también pueden estar en desacuerdo. Un argumento $\langle \mathcal{A}, H\rangle$ consiste de un conjunto de razones tentativas[2]. Pero puede ocurrir que existan otros argumentos cuyas conclusiones sean opuestas a H, o, al menos, sus conclusiones sean, en algún sentido, contradictorias con H, debido a que ponen en tela de juicio alguna de las razones subyacentes en las que se basa el argumento. En el primer caso, se tiene lo que se conoce en la literatura de argumentación como ataque por refutación (o *rebuttal*) o ataque directo y en el segundo lo que se conoce como ataque por socavamiento (o undercutting) o ataque indirecto. Si el argumento atacante es mejor, bajo algún criterio particular, entonces se lo considera un derrotador del primer argumento. A continuación desarrollamos estas cuestiones con el siguiente conjunto de definiciones.

Definición 3.3.11 (Ataque o contraargumento) *Se dice que $\langle \mathcal{A}_1, H_1\rangle$ contraargumenta (o ataca) a $\langle \mathcal{A}_2, H_2\rangle$ si y sólo si existe un subargumento $\langle \mathcal{A}, H\rangle$ de $\langle \mathcal{A}_2, H_2\rangle$ tal que H y H_1 están en desacuerdo.*

En la definición previa, el literal H es llamado punto de argumentación. Si H y H_1 son complementarios, entonces se está en el caso del ataque directo, sino, se está en el caso

[2]Excepto que todas las razones A consistan de reglas estrictas, en cuyo caso, el argumento $\langle \mathcal{A}, H\rangle$ constituiría una demostración de H en el sentido de la lógica clásica de primer orden o al menos en el sentido de la programación en lógica extendida tradicional.

de un ataque indirecto.

Consideremos los siguiente ejemplos que ilustran la noción de ataque.

Ejemplo 3.3.9 *Consideremos nuevamente el programa DeLP* $\mathcal{P}_{3.3.4} = (\Pi, \Delta)$ *presentado en el Ejemplo 3.3.4 de la página 69. En este programa, tenemos que el argumento* $\langle \mathcal{A}_1, flies(tweety) \rangle$ *no es atacado por ningún otro argumento. Por otro lado, el argumento* $\langle \mathcal{B}_1, flies(opus) \rangle$ *es atacado por el argumento* $\langle \mathcal{B}_2, \sim flies(opus) \rangle$ *y viceversa. Análogamente, lo mismo ocurre entre los argumentos* $\langle \mathcal{B}_2, \sim flies(opus) \rangle$ *y* $\langle \mathcal{B}_3, flies(opus) \rangle$*. En ambos casos, estamos en presencia de ataques directos.*

Ejemplo 3.3.10 *Revisemos nuevamente el programa DeLP* $\mathcal{P}_{3.3.8} = (\Pi, \Delta)$ *introducido en el Ejemplo 3.3.8 ubicado en la página 73. En este caso, el argumento* $\langle \mathcal{B}_2, \sim studies(paul) \rangle$ *ataca indirectamente al argumento* $\langle \mathcal{A}_1, pass(john) \rangle$*. En este caso, el punto de ataque es el literal* $studies(john)$ *dentro de la derivación del literal* $pass(john)$*. Así, los literales* $studies(john)$ *y* $\sim studies(john)$ *están en desacuerdo, ya que* $\Pi \cup \{studies(john), \sim studies(john)\}$ *es un conjunto trivialmente contradictorio.*

Se debe notar que, en el caso de un argumento $\langle \emptyset, H \rangle$, éste no tiene atacantes ya que, dicho argumento, al no hacer uso de razones tentativas, indica que H se deriva a utilizando exclusivamente reglas estrictas.

Ejemplo 3.3.11 *Considerando el programa DeLP* $\mathcal{P}_{3.3.8} = (\Pi, \Delta)$ *del Ejemplo 3.3.8 de la página 73, el argumento* $\langle \emptyset, uses\_browser(mary) \rangle$ *no tiene atacantes. La derivación correspondiente a este argumento se presenta en la Figura 3.7 de la página 76.*

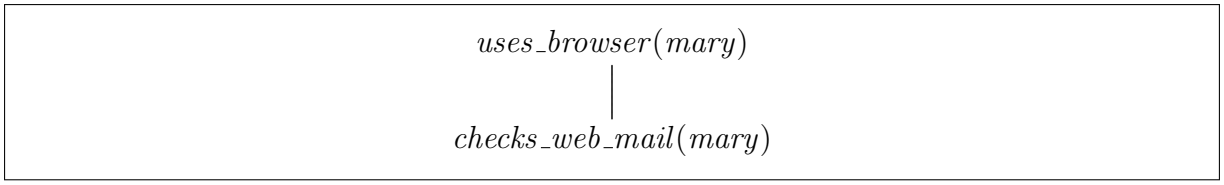


Figura 3.7: Derivación correspondiente al argumento $\langle \emptyset, uses\_browser(mary) \rangle$

En (García and Simari, 2004, Sección 6.1) se presenta un análisis detallado del uso de la negación por falla como una extensión manejable naturalmente en DeLP. Otros formalismos argumentativos (Chesñevar, Maguitman and Loui, 2000) hacen un uso análogo de este tipo de negación. Cuando se permite que las reglas contengan negación por falla, la noción de ataque se puede definir como sigue:

Definición 3.3.12 (Contraargumento. Derrota) *Un argumento* $\langle \mathcal{A}_1, Q_1 \rangle$ *es un contraargumento para un argumento* $\langle \mathcal{A}_2, Q_2 \rangle$ *si y sólo si :*

1. *Existe un subargumento* $\langle \mathcal{A}, Q \rangle$ *de* $\langle \mathcal{A}_2, Q_2 \rangle$ *tal que el conjunto* $\Pi \cup \{Q_1, Q\}$ *es contradictorio.*

2. *Un literal* **not** Q_1 *está presente en el cuerpo de alguna regla en* $\mathcal{A}_2$.

La primera noción de ataque es prestada del framework de Simari-Loui (Simari and Loui, 1992); la segunda está relacionada con el enfoque argumentativo de Dung a la programación en lógica (Dung, 1993*a*,*b*; Chesñevar, Maguitman and Loui, 2000) también como con otras formalizaciones, como por ejemplo (Prakken and Sartor, 1996; Bondarenko et al., 1997).

3.3.4. Comparación entre argumentos

Debido a que, dado un DeLP $\mathcal{P}$, el mismo puede contener información inconsistente (es decir, que permita derivar literales complementarios), debe existir un criterio que permita decidir cuál de dos argumentos en conflicto es mejor. En la literatura se pueden hallar al menos dos criterios para decidir qué argumento es mejor que otro: uno está basado en asignar *prioridades* a los argumentos basado en una asignación previa de prioridades a las reglas que lo conforman y el otro es llamado *especificidad*, basado en la forma que tienen los argumentos involucrados.

A continuación, describiremos estos dos enfoques para resolver qué argumento es mejor que otro cuando existen dos argumentos en conflicto.

Comparación de argumentos por prioridad de reglas

Algunos formalismos definen prioridades explícitas entre reglas y utilizan esas prioridades para decidir entre argumentos en conflicto. En dichos formalismos, la noción de derivación está ligada a un único criterio de comparación.

En la Programación en Lógica Rebatible, el criterio de comparación entre reglas es modular y puede ser reemplazado. García y Simari (García and Simari, 2004) definen un criterio de prioridad entre argumentos basados en una definición previa de prioridad entre las reglas de un programa lógico rebatible.

Definición 3.3.13 (Prioridad entre reglas) *Dado un programa lógico rebatible* P *y una relación de prioridad "*$\succ$*" definida explícitamente entre las reglas rebatibles. Dadas dos estructuras de argumentos* $\langle \mathcal{A}_1, H_1 \rangle$ *y* $\langle \mathcal{A}_2, H_2 \rangle$*, el argumento* $\langle \mathcal{A}_1, H_1 \rangle$ *será preferido sobre* $\langle \mathcal{A}_2, H_2 \rangle$ *si:*

1. *existe al menos una regla* $r_a \in \mathcal{A}_1$ *y una regla* $r_b \in A_2$*, tal que* $r_a \succ r_b$*, y,*

2. *no hay ninguna* $r'_b \in \mathcal{A}_2$ *ni* $r'_a \in \mathcal{A}_1$*, tal que* $r'_b \succ r'_a$.

Ejemplo 3.3.12 *Sea* $\mathcal{P}_{3.3.12} = (\Pi, \Delta)$ *el siguiente programa DeLP con:*

$$\begin{array}{rcl} \Pi & = & \left\{ \begin{array}{l} bird(tweety); \\ penguin(opus) \end{array} \right\} y \\ \Delta & = & \left\{ \begin{array}{l} bird(X) \prec penguin(X); \\ flies(X) \prec bird(X); \\ \sim flies(X) \prec penguin(X) \end{array} \right\}. \end{array}$$

Sea la siguiente prioridad entre reglas tal que:

$$\left(\sim flies(X) \prec penguin(X) \right) \succ \left(flies(X) \prec bird(X) \right).$$

A partir de este programa tenemos dos argumentos en conflicto $\langle \mathcal{A}_1, flies(opus) \rangle$ *y* $\langle \mathcal{A}_1, \sim flies(opus) \rangle$*, donde:*

$$\begin{array}{rcl} \mathcal{A}_1 & = & \left\{ \begin{array}{l} flies(opus) \prec bird(opus); \\ bird(opus) \prec penguin(opus) \end{array} \right\}, y \\ \mathcal{A}_2 & = & \left\{ \sim flies(opus) \prec penguin(opus) \right\}. \end{array}$$

De acuerdo al criterio de comparación de reglas, el argumento $\mathcal{A}_2$ *derrota al argumento* $\mathcal{A}_1$*. Note que existe un tercer argumento* $\langle \mathcal{A}_3, flies(tweety) \rangle$*, con*

$$\mathcal{A}_3 \quad = \quad \left\{ flies(tweety) \prec bird(tweety) \right\}$$

que no es derrotado por ningún argumento.

Comparación de argumentos por especificidad

La noción de especificidad entre argumentos es un criterio que sirve para discriminar entre dos argumentos en conflicto. Intuitivamente, esta noción favorece dos aspectos en un argumento: prefiere un argumento con (1) más contenido de información o (2) más directo (con menos uso de reglas). Es decir, un argumento será mejor que otro si es más preciso o más conciso.

El primer caso habla de argumentos cuyas reglas tienen más antecedentes; por lo tanto, se puede decir que para utilizarlo hubo que tener más información a la mano (para poder probar los antecedentes). En el segundo caso, debido a que las reglas rebatibles son tentativas, su uso introduce en cierto modo incertidumbre respecto de la conclusión final; por lo tanto, cuánto más reglas de este tipo se hubieran usado, más incierta es la conclusión obtenida.

Ahora, introducimos la definición de especificidad mencionada en (García and Simari, 2004) pero cuya introducción en la literatura se remonta al trabajo de Poole y al trabajo seminal de Simari (Simari and Loui, 1992).

Definición 3.3.14 (Especificidad entre argumentos) *Sea $P = (\Pi, \Delta)$ un programa lógico rebatible y sea Π_G el conjunto de todas las reglas estrictas de Π (sin incluir los hechos). Sea F el conjunto de todos los literales que tienen una derivación rebatible a partir de P (F será considerado como un conjunto de hechos). Sean $\langle \mathcal{A}_1, H_1 \rangle$ y $\langle \mathcal{A}_2, H_2 \rangle$ dos estructuras de argumentos obtenidas a partir de P. $\langle \mathcal{A}_1, H_1 \rangle$ es estrictamente más específico que $\langle \mathcal{A}_2, H_2 \rangle$ (denotado como $\langle \mathcal{A}_1, H_1 \rangle \succ \langle \mathcal{A}_2, H_2 \rangle$) si se cumplen las siguientes condiciones:*

1. *Para todo $H \subseteq F$: si $\Pi_G \cup H \cup \mathcal{A}_1 \vdash H_1$ y $\Pi_G \cup H \not\vdash H_1$, entonces $\Pi_G \cup H \cup \mathcal{A}_2 \vdash H_2$, y,*
2. *existe $H' \subseteq F$ tal que $\Pi_G \cup H' \cup \mathcal{A}_2 \vdash H_2$ y $\Pi_G \cup H' \not\vdash H_2$, y $\Pi_G \cup H' \cup \mathcal{A}_1 \not\vdash H_1$.*

Como hemos mencionado más arriba, intuitivamente, la comparación de argumentos por especificidad se basa en dos conceptos: directitud y mayor información. La *directitud* es el dual de la cantidad de pasos de inferencia necesarios para obtener un conclusión. La *mayor información* se basa en la cantidad de premisas que apoyan a una conclusión.

La noción de comparación de argumentos por directitud se apoya en varias intuiciones. Cuanto menos pasos de inferencia involucra la derivación de un argumento, éste se dice que es "más directo". Debido a que la utilización de reglas rebatibles en la derivación de argumentos implica la introducción de un cierto grado de incertidumbre en el proceso de razonamiento rebatible. A mayor cantidad de pasos de derivación rebatibles utilizados, mayor es la duda existente sobre la conclusión final obtenida. Así, si tuviéramos dos argumentos contradictorios, preferimos, de acuerdo al criterio de directitud, a aquel que involucre menos pasos de derivación rebatibles. Consideremos estas nociones en el siguiente ejemplo.

Ejemplo 3.3.13 *Consideremos el siguiente programa DeLP proposicional:*

$$\mathcal{P}_4 = \left\{ \begin{array}{l} c \prec b; \\ b \prec a; \\ \sim c \prec a; \\ a \end{array} \right\}.$$

De acuerdo a la regla de especificidad, el argumento $\langle \{(c \prec b); (b \prec c)\}, c \rangle$ tendrá menos precedencia que el argumento $\langle \{\sim c \prec a\}, \sim c \rangle$.

También la noción de comparación por mayor información se apoya en varias intuiciones. La credibilidad de un argumento crece a medida que crece el número de hipótesis que lo soportan. Es decir, dados dos argumentos contradictorios, de acuerdo al criterio de mayor información, preferiremos a aquel que posee más premisas.

Para soportar este enfoque, consideremos que tomamos una decisión en base a pruebas empíricas o experimentos realizados previamente; claramente, aquella decisión basada

en más experimentos es la más creible. En el caso de los argumentos, los experimentos vienen a ser las premisas y la decisión, la conclusión. Entonces, en el caso de la comparación de argumentos por mayor información, el resultado basado en la mayor cantidad de experimentos (o premisas) es el más confiable. Veamos un ejemplo.

Ejemplo 3.3.14 *Consideremos el siguiente clásico programa DeLP proposicional:*

$$\mathcal{P}_5 = \left\{ \begin{array}{l} c \prec a, b \\ \sim c \prec a \\ a \\ b \end{array} \right\}.$$

De acuerdo a la regla de especificidad, el argumento $\langle \{c \prec a, b\}, c \rangle$ *tendrá más precedencia que el argumento* $\langle \{\sim c \prec a\}, \sim c \rangle$.

3.3.5. Garantía como análisis dialéctico en DeLP

Hasta ahora, vimos que dos argumentos están en desacuerdo si ambos no pueden ser aceptados al mismo tiempo debido a que ello produciría una inconsistencia con respecto a la base de creencias estrictas. También, vimos que un argumento ataca a otro si el primero está en desacuerdo con alguno de las conclusiones (intermedias o finales) en las que se apoya el segundo. Además, vimos que un argumento derrota a otro si lo ataca y es mejor respecto de algún criterio de comparación de argumentos previamente establecido.

Sin embargo, un argumento derrotado puede ser reinstaurado hallando un tercer argumento, que constituya un derrotador para el argumento que lo destituyó. Pero, a su vez, este tercer argumento puede ser derrotado por un cuarto, quien reinstauraría al derrotador original. Este proceso, llamado *argumentación*, puede ser visto como un proceso en el que dos partes, *proponente* y *oponente*, dirimen la cuestión planteada por el argumento original. Como en los debates de la vida real, aquella parte que tiene la última palabra (es decir cuyo último argumento no puede ser derrotado) se considera la parte triunfadora en la discusión dialéctica. La secuencia de argumentos a favor y en contra se llama *línea de argumentación*. Es deseable también que, en una línea de argumentación, todos los argumentos a favor del argumento original y todos los argumentos en contra de éste sean consistentes entre sí (lo que se ha dado en llamar *concordancia*). Además, un argumento usado previamente no puede introducirse nuevamente en una línea de argumentación (esto evita la falacia de la demostración circular o *circulus in demonstrando*).

Sin embargo, a partir de un argumento inicial a favor de una conclusión, pueden plantearse varias líneas de argumentación. El objetivo último consiste en determinar si un argumento dado puede salir victorioso en todas las líneas de argumentación.

A partir de estas premisas de trabajo, el trabajo de García (García, 1997) consistió en sistematizar este proceso de búsqueda de líneas de argumentación a partir de un

argumento inicial a favor de un literal dado. Así el intérprete DeLP implementado por García explora con una estrategia en profundidad (en inglés, *depth-first*) todas las líneas de argumentación en contra de dicho argumento. El espacio de búsqueda constituido por las líneas de argumentación exploradas conforma un árbol de dialéctica, concepto expuesto previamente en la tesis doctoral de Guillermo Simari (Simari, 1989; Simari and Loui, 1992)), en el que las hojas están conformadas por argumentos no derrotados.

También, se han propuesto trabajos en el cual se poda el recorrido de dicho espacio de búsqueda siguiendo una heurística similar al podado α-β (en inglés, α-β *pruning*) (Chesñevar, Simari and García, 2000). Así, para determinar si un argumento se halla garantizado no es necesario explorar todo el árbol de dialéctica, haciendo más eficiente el proceso.

A continuación, daremos las definiciones formales del proceso resumidamente descripto.

Derrotadores

Dada una estructura de argumento $\langle \mathcal{A}_1, H_1 \rangle$ y un contraargumento $\langle \mathcal{A}_2, H_2 \rangle$ para $\langle \mathcal{A}_1, H_1 \rangle$, estos dos argumentos se pueden comparar para decidir cuál de ellos prevalece. Si el contraargumento $\langle \mathcal{A}_2, H_2 \rangle$ es mejor que $\langle \mathcal{A}_1, H_1 \rangle$ con respecto al criterio de comparación, entonces $\langle \mathcal{A}_2, H_2 \rangle$ es un *derrotador propio* para A_1. Si ningún argumento es mejor o peor que el otro (pues el orden entre argumentos es parcial), se dice que ocurre una situación de bloqueo y $\langle \mathcal{A}_2, H_2 \rangle$ es un derrotador *por bloqueo* para $\langle \mathcal{A}_1, H_1 \rangle$. Si $\langle \mathcal{A}_1, H_1 \rangle$ es mejor que $\langle \mathcal{A}_2, H_2 \rangle$, entonces $\langle \mathcal{A}_2, H_2 \rangle$ no es considerado como derrotador de $\langle \mathcal{A}_1, H_1 \rangle$.

Siguiendo la línea de Simari y García, en lo que sigue se denotará la relación de orden parcial entre predicados como $\succ$.

Definición 3.3.15 (Derrotador propio) *Sean $\langle \mathcal{A}_1, H_1 \rangle$ y $\langle \mathcal{A}_2, H_2 \rangle$ dos estructuras de argumento. $\langle \mathcal{A}_1, H_1 \rangle$ es un derrotador propio de $\langle \mathcal{A}_2, H_2 \rangle$ en el literal H, si y sólo si existe un subargumento $\langle \mathcal{A}, H \rangle$ de $\langle \mathcal{A}_2, H_2 \rangle$ tal que $\langle \mathcal{A}_1, H_1 \rangle$ contraargumenta $\langle \mathcal{A}_2, H_2 \rangle$ en H y $\langle \mathcal{A}_1, H_1 \rangle \succ \langle \mathcal{A}, H \rangle$.*

Definición 3.3.16 (Derrotador por bloqueo) *Sean $\langle \mathcal{A}_1, H_1 \rangle$ y $\langle \mathcal{A}_2, H_2 \rangle$ dos estructuras de argumento. $\langle \mathcal{A}_1, H_1 \rangle$ es un derrotador por bloqueo de $\langle \mathcal{A}_2, H_2 \rangle$ en el literal H, si y sólo si existe un subargumento $\langle \mathcal{A}, H \rangle$ de $\langle \mathcal{A}_2, H_2 \rangle$ tal que $\langle \mathcal{A}_1, H_1 \rangle$ contraargumenta $\langle \mathcal{A}_2, H_2 \rangle$ en H y $\langle \mathcal{A}_1, H_1 \rangle$ no está relacionado por el orden de preferencia a $\langle \mathcal{A}, H \rangle$, es decir, $\langle \mathcal{A}_1, H_1 \rangle \not\succ \langle \mathcal{A}, H \rangle$ ni $\langle \mathcal{A}, H \rangle \not\succ \langle \mathcal{A}_1, H_1 \rangle$.*

Definición 3.3.17 (Derrotador) *La estructura de argumento $\langle \mathcal{A}_1, H_1 \rangle$ es un derrotador para $\langle \mathcal{A}_2, H_2 \rangle$, si y sólo si ocurre una de dos cosas: (1) $\langle \mathcal{A}_1, H_1 \rangle$ es un derrotador propio de $\langle \mathcal{A}_2, H_2 \rangle$, o, (2) $\langle \mathcal{A}_1, H_1 \rangle$ es un derrotador por bloqueo de $\langle \mathcal{A}_2, H_2 \rangle$.*

Para ver cómo se aplican estos conceptos a los argumentos presentados en las definiciones anteriores del marco DeLP, veamos los siguientes ejemplos.

Ejemplo 3.3.15 *Consideremos nuevamente el programa DeLP* $\mathcal{P}_{3.3.4} = (\Pi, \Delta)$ *presentado en el Ejemplo 3.3.4 de la página 69. Al utilizar el criterio de especificidad generalizada en tal programa DeLP, el argumento* $\langle \mathcal{B}_3, flies(opus) \rangle$ *derrota al argumento* $\langle \mathcal{B}_2, \sim flies(opus) \rangle$ *ya que el primero está más informado.*

Ejemplo 3.3.16 *Veamos nuevamente el programa DeLP* $\mathcal{P}_{3.3.8} = (\Pi, \Delta)$ *del Ejemplo 3.3.8 en la página 73. En dicho programa, además de los argumentos presentados, tenemos otro* $\langle \mathcal{A}_3, pass(paul) \rangle$ *donde*

$$\mathcal{A}_3 = \left\{ \begin{array}{l} pass(paul) \prec studies(paul); \\ studies(paul) \prec seats\_in\_computer(paul) \end{array} \right\}.$$

Utilizando el criterio de especificidad, el argumento $\mathcal{A}_3$ *es derrotado por el argumento* $\mathcal{A}_2$ *presentado en el Ejemplo 3.3.8.*

Líneas de argumentación

Dado un argumento original, se puede hallar un derrotador para éste; luego, otro derrotador para el segundo, que reinstaure al primero; después un cuarto que derrote al tercero y nuevamente deponga al primero y así sucesivamente.

Definición 3.3.18 (Línea de argumentación) *Sea* $\mathcal{P}$ *un DeLP y* $\langle \mathcal{A}_0, H_0 \rangle$ *una estructura de argumento obtenida a partir de* $\mathcal{P}$. *Una línea de argumentación para* $\langle \mathcal{A}_0, H_0 \rangle$ *es una secuencia de estructuras de argumentos a partir de* $\mathcal{P}$, *denotada* $\Lambda = [\langle \mathcal{A}_0, H_0 \rangle, \langle \mathcal{A}_1, H_1 \rangle, \langle \mathcal{A}_2, H_2 \rangle, \langle \mathcal{A}_3, H_3 \rangle, \ldots]$, *donde cada elemento de la secuencia* $\langle \mathcal{A}_i, H_i \rangle$, $i > 0$, *es un derrotador para su predecesor* $\langle \mathcal{A}_{i-1}, H_{i-1} \rangle$.

De esta manera, los argumentos pares se dice que soportan la conclusión original mientras que los impares la interfieren.

Definición 3.3.19 (Argumentos de soporte e interferencia) *Sea una línea de argumentación* $\Lambda = [\langle \mathcal{A}_0, H_0 \rangle, \langle \mathcal{A}_1, H_1 \rangle, \langle \mathcal{A}_2, H_2 \rangle, \langle \mathcal{A}_3, H_3 \rangle, \ldots]$, *se define el conjunto de estructuras de argumento de soporte como* $\Lambda_S = \{\langle \mathcal{A}_0, H_0 \rangle, \langle \mathcal{A}_2, H_2 \rangle, \langle \mathcal{A}_4, H_4 \rangle, \ldots\}$ *y el conjunto de estructuras de argumento de interferencia como* $\Lambda_I = \{\langle \mathcal{A}_1, H_1 \rangle, \langle \mathcal{A}_3, H_3 \rangle, \langle \mathcal{A}_5, H_5 \rangle, \ldots\}$.

Tanto los argumentos de soporte como los de interferencia en una línea de argumentación dada deben cumplir ciertas propiedades de coherencia entre sí. Una de ellas es la concordancia.

Definición 3.3.20 (Concordancia) *Sea $\mathcal{P} = (\Pi, \Delta)$ un programa DeLP. Dos argumentos $\langle \mathcal{A}_1, H_1 \rangle$ y $\langle \mathcal{A}_2, H_2 \rangle$ son concordantes si y sólo si el conjunto $\Pi \cup \mathcal{A}_1 \cup \mathcal{A}_2$ es no contradictorio. Más generalmente, un conjunto de estructuras de argumento $\{\langle \mathcal{A}_i, H_i \rangle : i = 1, \ldots, n\}$ es concordante si y sólo si $\Pi \cup \bigcup_1^n \mathcal{A}_i$ es no contradictorio.*

Definición 3.3.21 (Línea de argumentación aceptable) *Sea una línea de argumentación $\Lambda = [\langle \mathcal{A}_1, H_1 \rangle, \langle \mathcal{A}_2, H_2 \rangle, \ldots, \langle \mathcal{A}_n, H_n \rangle]$. Λ es una línea de argumentación aceptable si y sólo si:*

1. *Λ es una secuencia finita.*
2. *El conjunto Λ_S, de argumentos de soporte, es concordante y el conjunto Λ_I, de argumentos de interferencia, es concordante.*
3. *Ningún argumento $\langle \mathcal{A}_k, H_k \rangle$ en Λ es un subargumento de un argumento $\langle \mathcal{A}_i, H_i \rangle$ apareciendo previamente en Λ $(i < k)$.*
4. *Para todo i, tal que el argumento $\langle \mathcal{A}_i, H_i \rangle$ es un derrotador por bloqueo de $\langle \mathcal{A}_{i-1}, H_{i-1} \rangle$, si existe $\langle \mathcal{A}_{i+1}, H_{i+1} \rangle$, entonces $\langle \mathcal{A}_{i+1}, H_{i+1} \rangle$ es un derrotador propio para $\langle \mathcal{A}_i, H_i \rangle$.*

Ejemplo 3.3.17 *Veamos nuevamente el programa DeLP $\mathcal{P}_{3.3.4} = (\Pi, \Delta)$ presentado en el Ejemplo 3.3.4 de la página 69. Tres de los argumentos que se podían construir a partir de dicho programa fueron presentados en el Ejemplo 3.3.7 de la página 72: $\langle \mathcal{B}_1, flies(opus) \rangle$, $\langle \mathcal{B}_2, \sim flies(opus) \rangle$ y $\langle \mathcal{B}_3, flies(opus) \rangle$, con:*

$$\begin{array}{rcl} \mathcal{B}_1 & = & \left\{ \; flies(opus) \prec bird(opus) \; \right\}; \\ \mathcal{B}_2 & = & \left\{ \; \sim flies(opus) \prec penguin(opus) \; \right\}, \text{ y,} \\ \mathcal{B}_3 & = & \left\{ \; flies(opus) \prec penguin(opus), owns(opus, cbd05), plane(cbd05) \; \right\}. \end{array}$$

Una línea posible de argumentación se muestra en la Figura 3.8 de la página 84. En la misma y recurriendo al principio de especificidad para comparar argumentos, el argumento $\mathcal{B}_1$ es derrotado propiamente por el argumento $\mathcal{B}_2$, quien, a su vez, es derrotado propiamente por el argumento $\mathcal{B}_3$. Entonces, en este caso, $\mathcal{B}_1$ y $\mathcal{B}_3$ son considerados los argumentos de soporte mientras que $\mathcal{B}_2$ es el argumento de interferencia.

Ejemplo 3.3.18 *Consideremos nuevamente el programa DeLP $\mathcal{P}_{3.3.8} = (\Pi, \Delta)$ introducido en el Ejemplo 3.3.8 ubicado en la página 73. Tenemos argumentos respecto a la situación de Mary que no hemos considerado previamente:*

- *$\langle \mathcal{C}_1, pass(mary) \rangle$, con:*

$$\mathcal{C}_1 = \left\{ \begin{array}{l} pass(mary) \prec studies(mary); \\ studies(mary) \prec seats\_in\_computer(mary) \end{array} \right\}$$

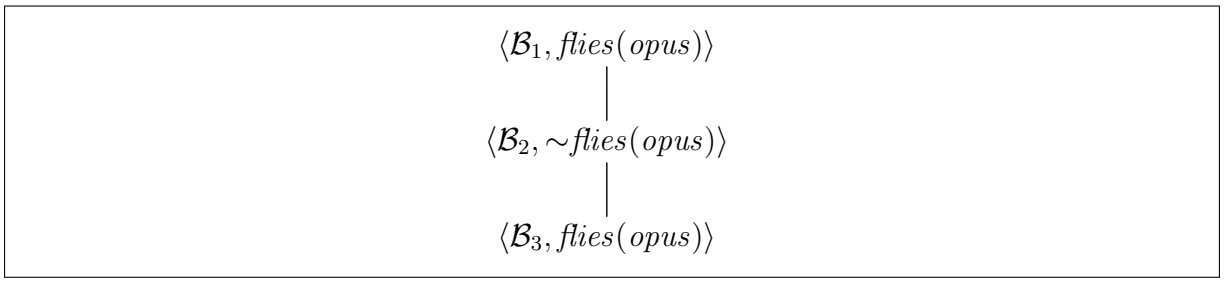


Figura 3.8: Línea de argumentación posible para el programa DeLP $\mathcal{P}_{3.3.4}$

- $\langle \mathcal{C}_2, \sim pass(mary) \rangle$, *con:*

$$\mathcal{C}_2 = \left\{ \begin{array}{l} \sim pass(mary) \prec \sim studies(mary); \\ \sim studies(mary) \prec seats\_in\_computer(mary), web\_surfing(mary); \\ web\_surfing(mary) \prec uses\_browser(mary) \end{array} \right\}$$

- $\langle \mathcal{C}_3, \sim web\_surfing(mary) \rangle$, *con:*

$$\mathcal{C}_3 = \left\{ \sim web\_surfing(mary) \prec uses\_browser(mary), reads\_javadoc(mary) \right\}$$

- $\langle \mathcal{C}_4, \sim studies(mary) \rangle$, *con:*

$$\mathcal{C}_4 = \left\{ \begin{array}{l} \sim studies(mary) \prec seats\_in\_computer(mary), web\_surfing(mary); \\ web\_surfing(mary) \prec uses\_browser(mary) \end{array} \right\}.$$

Dos posibles líneas de argumentación se muestran en la Figura 3.9. En la línea de argumentación (a), el argumento $\mathcal{C}_2$ derrota por bloqueo a $\mathcal{C}_1$ tomando como punto de ataque a su conclusión misma; $\mathcal{C}_2$ es derrotado propiamente por $\mathcal{C}_3$ usando al literal $web\_surfing(mary)$ como punto de ataque. En la línea de argumentación (b), el argumento $\mathcal{C}_1$ es derrotado propiamente en el literal $studies(mary)$ por el argumento $\mathcal{C}_4$, quien a su vez es también derrotado propiamente por el argumento $\mathcal{C}_3$ en el literal $web\_surfing(mary)$.

Análisis dialéctico

Como cada argumento puede tener varios derrotadores, a partir de cada argumento se pueden iniciar una variedad de líneas de argumentación. Esta idea da lugar al árbol dialéctico.

En DeLP, un literal H se dice garantizado si existe una estructura de argumento $\langle \mathcal{A}, H \rangle$ no derrotada. Para establecer si $\langle \mathcal{A}, H \rangle$ no está derrotado, se considera el conjunto de derrotadores para $\langle \mathcal{A}, H \rangle$. En virtud de que cada derrotador D para $\mathcal{A}$ es una estructura de argumento, se consideran también los argumentos para D y así sucesivamente. Luego, puede surgir más de una línea de argumentación, llevando a la aparición de una estructura arbórea llamada árbol dialéctico.

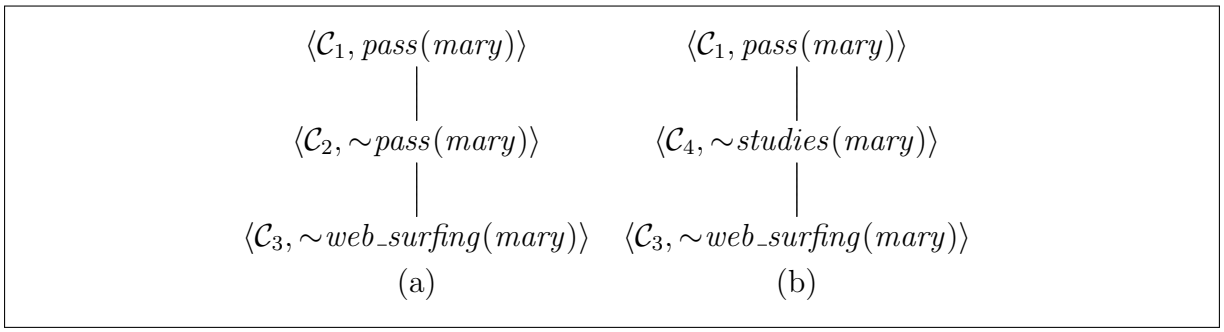


Figura 3.9: Dos posibles líneas de argumentación para el programa DeLP $\mathcal{P}_{3.3.8}$

Definición 3.3.22 (Árbol dialéctico) *Sea $\langle \mathcal{A}_0, H_0 \rangle$ una estructura de programa construída a partir de un programa P. Un árbol dialéctico para $\langle \mathcal{A}_0, H_0 \rangle$, denotado como $\mathfrak{Tree}_{\langle \mathcal{A}_0, H_0 \rangle}$, se define como:*

1. *La raíz del árbol se halla etiquetada con $\langle \mathcal{A}_0, H_0 \rangle$.*

2. *Sea N un nodo interno del árbol etiquetado como $\langle \mathcal{A}_n, H_n \rangle$ y $\Lambda = [\langle \mathcal{A}_0, H_0 \rangle, \langle \mathcal{A}_1, H_1 \rangle, \ldots, \langle \mathcal{A}_n, H_n \rangle]$ la secuencia de las etiquetas de los caminos a partir del nodo N. Sean $\langle \mathcal{B}_1, Q_1 \rangle$, $\langle \mathcal{B}_2, Q_2 \rangle$, $\ldots$, $\langle \mathcal{B}_k, Q_k \rangle$ todos los derrotadores para $\langle \mathcal{A}_n, H_n \rangle$. Para cada derrotador $\langle \mathcal{B}_i, Q_i \rangle$ $(i = 1, \ldots, k)$, tal que, la línea de argumentación $\Lambda' = [\langle \mathcal{A}_0, H_0 \rangle, \langle \mathcal{A}_1, H_1 \rangle, \langle \mathcal{A}_2, H_2 \rangle, \ldots, \langle \mathcal{A}_n, H_n \rangle, \langle \mathcal{B}_i, Q_i \rangle]$ es aceptable, entonces el nodo N tiene un hijo N_i etiquetado como $\langle \mathcal{B}_i, Q_i \rangle$. Si no existe ningún derrotador para $\langle \mathcal{A}_n, H_n \rangle$ o no existe ningún $\langle \mathcal{B}_i, Q_i \rangle$ tal que Λ' sea aceptable, entonces N es una hoja.*

En un árbol dialéctico, cada nodo (excepto la raíz) representa un derrotador (propio o por bloqueo) de su padre y las hojas corresponden a argumentos no derrotados. Cada camino desde la raíz hacia una hoja corresponde a una línea de argumentación aceptable diferente. En resumen, el árbol dialéctico representa un análisis dialéctico exhaustivo para el argumento en su raíz.

Para decidir si el argumento en la raíz de un árbol dialéctico está derrotado, se define un proceso de marcado del árbol dialéctico. Los nodos se marcan como *"D"* (*derrotado*) o *"U"* (*no derrotado*) como sigue:

Definición 3.3.23 (Marcado del árbol dialéctico) *Sea $\mathfrak{Tree}_{\langle \mathcal{A}, H \rangle}$ un árbol dialéctico para $\langle \mathcal{A}, H \rangle$. El árbol dialéctico marcado correspondiente, denotado como $\mathfrak{Tree}^*_{\langle \mathcal{A}, H \rangle}$, se obtiene marcando cado nodo de $\mathfrak{Tree}_{\langle \mathcal{A}, H \rangle}$ como sigue:*

1. *Todas las hojas en $\mathfrak{Tree}_{\langle \mathcal{A}, H \rangle}$ se marcan como U en $\mathfrak{Tree}^*_{\langle \mathcal{A}, H \rangle}$.*

2. *Sea* $\langle \mathcal{B}, Q \rangle$*un nodo interno de* $\mathfrak{Tree}_{\langle \mathcal{A},H \rangle}$*. Entonces,* $\langle \mathcal{B}, Q \rangle$*se marca como* U *en* $\mathfrak{Tree}^*_{\langle \mathcal{A},H \rangle}$ *si y sólo si cada hijo de* $\langle \mathcal{B}, Q \rangle$*es marcado como* D*. El nodo* $\langle \mathcal{B}, Q \rangle$*se marca como* D *si y sólo si tiene al menos un hijo marcado como* U*.*

La noción de garantía se define a partir de un árbol dialéctico marcado como sigue:

Definición 3.3.24 (Literales garantizados) *Sea* $\langle \mathcal{A}, H \rangle$*una estructura de argumento y* $\mathfrak{Tree}^*_{\langle \mathcal{A},H \rangle}$ *su árbol dialéctico marcado asociado. El literal* H *está garantizado si y sólo si la raíz de* $\mathfrak{Tree}^*_{\langle \mathcal{A},H \rangle}$ *está marcada como* U*. Diremos que* A *es una garantía para* H*.*

Definición 3.3.25 (Respuestas a consultas (García and Simari, 2004)) *Dado un programa DeLP* $\mathcal{P} = (\Pi, \Delta)$ *y dada un consulta (literal)* Q*, existen cuatro posibles respuestas para una consulta* Q *respecto de* $\mathcal{P}$*:*

- Sí*, corresponde a creer en* Q *si y sólo si existe un argumento garantizado a favor de* Q*;*
- No*, corresponde a creer en* $\overline{Q}$ *si y sólo si existe un argumento garantizado a favor de* $\overline{Q}$*;*
- Indeciso*, si y sólo si* Q *ni* $\sim Q$ *no están garantizados, y,*
- Desconocido*, si* Q *no pertenece al lenguaje del programa* $\mathcal{P}$*.*

Ejemplo 3.3.19 *Respecto del programa DeLP* $\mathcal{P}_{3.3.4} = (\Pi, \Delta)$ *presentado en el Ejemplo 3.3.4 de la página 69, cuando consideramos el problema de hallar la respuesta a la consulta* $flies(tweety)$*, debemos considerar marcado del árbol dialéctico presentado en la Figura 3.10.(a) en la página 87. En este caso, vemos que el árbol está compuesto por un único nodo marcado como* U*, indicando que dicho argumento no posee derrotadores. Así, la respuesta a la consulta es* Sí*.*

En el caso de la consulta $flies(opus)$*, vemos el árbol dialéctico en la Figura 3.10.(b). El argumento* $\mathcal{B}_1$ *aparece no derrotado y por lo tanto la respuesta a la consulta es también* Sí*. Por otro lado, si consideramos la consulta* $\sim flies(opus)$*, el árbol resultante es el de la Figura 3.10.(c), como su raíz está marcada como* D*, la respuesta a dicha consulta será* No*.*

Ejemplo 3.3.20 *Respecto de algunas de las consultas que podemos realizar respecto del programa* $\mathcal{P}_{3.3.8} = (\Pi, \Delta)$ *presentado en el Ejemplo 3.3.8 de la página 73, podemos considerar las siguientes:* $pass(mary)$*,* $pass(john)$ *y* $pass(paul)$*. Mostraremos por qué las respuestas a dichas consultas son* Sí*,* No *e* Indeciso*, respectivamente.*

Consideremos primero la consulta $pass(mary)$*. El árbol dialéctico para dicha consulta está formado por las dos líneas de argumentación presentadas previamente en el*

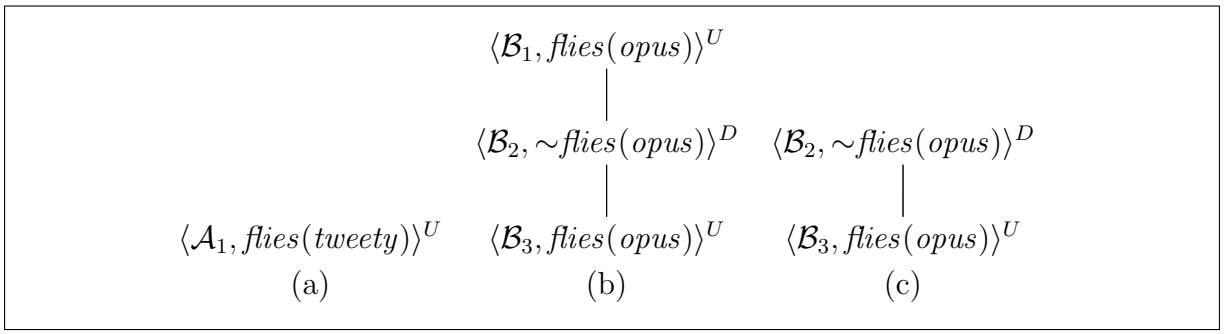


Figura 3.10: Árboles dialécticos para las consultas *flies*(*tweety*), *flies*(*opus*) y ∼*flies*(*opus*) respecto de $\mathcal{P}_{3.3.4}$

Ejemplo 3.3.18 de la página 83. Así, el árbol dialéctico para dicha consulta será el de la Figura 3.11 de la página 88.

En el caso de la consulta pass(*john*)*, para obtener la respuesta es necesario considera solamente el árbol compuesto por el argumento* $\langle \mathcal{D}_1, pass(john) \rangle$ *donde:*

$$\mathcal{D}_1 = \left\{ \begin{array}{l} pass(john) \prec studies(john); \\ studies(john) \prec seats\_in\_computer(john) \end{array} \right\}$$

Por lo tanto, el árbol de dialéctica tendrá un único argumento (no derrotado) y marcado como U. Así, el valor de la respuesta es Sí.

En el caso de la consulta pass(*paul*)*, debemos considerar varios argumentos y cómo los mismos se atacan entre sí:*

- *Tenemos un argumento* $\langle \mathcal{E}_1, \sim pass(paul) \rangle$ *que indica que Paul no aprobará pues no estudia porque está todo el día sentado en la computadora navegando por la web:*

$$\mathcal{E}_1 = \left\{ \begin{array}{l} \sim pass(paul) \prec \sim studies(paul); \\ \sim studies(paul) \prec seats\_in\_computer(paul), web\_surfing(paul); \\ web\_surfing(paul) \prec uses\_browser(paul) \end{array} \right\}$$

- *Hay otro argumento* $\langle \mathcal{E}_2, pass(paul) \rangle$ *que ataca por bloqueo a* $\mathcal{E}_1$*; este argumento dice que Paul estudia porque se sienta en la computadora y por ello aprobará el examen:*

$$\mathcal{E}_2 = \left\{ \begin{array}{l} pass(paul) \prec studies(paul); \\ studies(paul) \prec seats\_in\_computer(paul) \end{array} \right\}.$$

Debido a la forma simétrica del ataque entre estos dos argumentos, también se da el caso que el argumento $\mathcal{E}_1$ *ataca por bloqueo al argumento* $\mathcal{E}_2$*. Por lo tanto, la respuesta a la consulta pass*(*paul*) *es* Indeciso *y está basada en los dos árboles de dialéctica que se muestran en la Figura 3.12 de la página 88.*

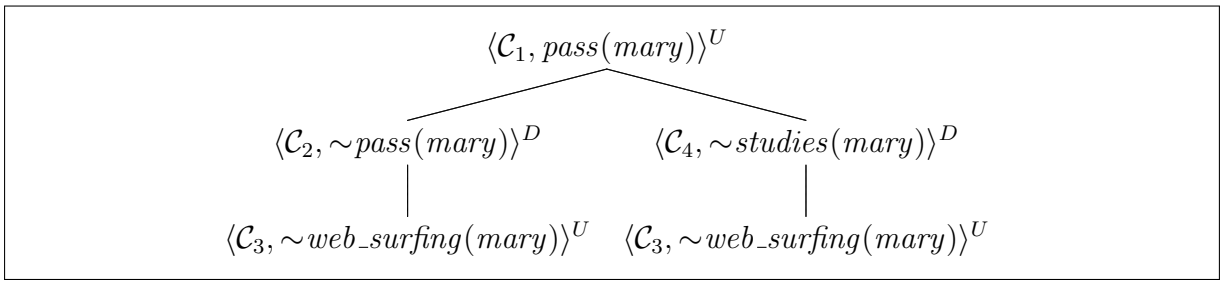


Figura 3.11: Árbol dialéctico para la consulta $pass(mary)$ respecto de $\mathcal{P}_{3.3.8}$

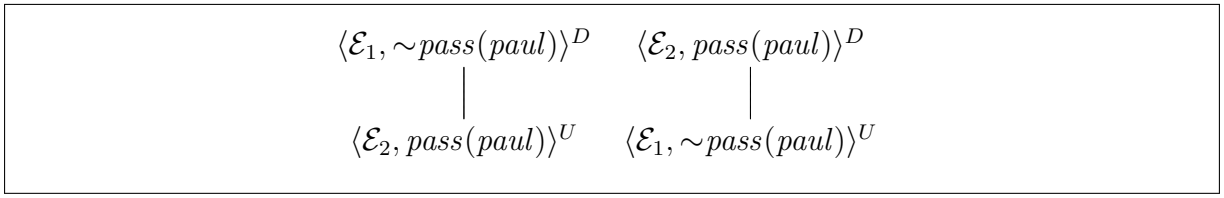


Figura 3.12: Árboles dialécticos para la consulta $pass(paul)$ respecto de $\mathcal{P}_{3.3.8}$

3.4. Resumen

La lógica clásica tiene serias restricciones a la hora de representar incompletitud e inconsistencia en la información. La argumentación rebatible ha surgido en los últimos años como un enfoque que permite lidiar con información incompleta y potencialmente contradictoria.

Luego, la argumentación provee una perspectiva diferente al razonamiento no-monótono y al razonamiento rebatible, en el cual una proclama es aceptada o rechazada en la base de los argumentos en favor o en contra de ella, y en el hecho de si dichos argumentos son atacados y derrotados por otros (Carbogim et al., 2000). Esta visión es soportada por la *argumentación rebatible*. Los marcos de argumentación tienen generalmente estos elementos: un lenguaje lógico subyacente, un concepto de argumento, un concepto de conflicto entre argumentos, una noción de derrota entre argumentos, y, una noción de cuándo un argumento es aceptable.

Las conclusiones tentativas se presentan en forma de argumentos soportados por un conjunto de premisas. Éstos a su vez son atacados y puestos a prueba por otro argumentos que apoyan a conclusiones contradictorias con las primeras. Luego se lleva a cabo un proceso en el que los mejores argumentos salen victoriosos. Si bien los sistemas presentados tienen diferencias entre sí, usualmente todos soportan las ideas previas en una u otra forma.

Vimos como la Programación en Lógica Rebatible es un sistema de argumentos con una implementación concreta y eficiente. En particular, la Programación en Lógica Rebatible se presenta como una herramienta muy potente para modelar situaciones en las que

se necesite computar el estado epistémico de una consulta respecto de una base de conocimientos tentativa. Así, mostraremos en capítulos posteriores cómo se puede integrar a la Programación en Lógica Rebatible con un formalismo de representación de ontologías para poder obtener respuestas significativas en presencia de bases de conocimiento incompletas y posiblemente inconsistentes.

Capítulo 4

Integración de ontologías y argumentación rebatible

La motivación de la Web Semántica es construir un espacio de representación de conocimiento que permita a los agentes inteligentes con poder delegado por sus usuarios ser capaces de comprender la información allí presente y, de esta manera, poder tomar decisiones a favor de dichos usuarios. La información en la Web Semántica es definida por medio de ontologías, que son formalizaciones del conocimiento de un dominio de aplicación. A pesar de que las definiciones de ontologías expresadas en Lógicas para la Descripción pueden ser procesadas por razonadores estándar, tales razonadores son incapaces de lidiar con ontologías inconsistentes. Investigaciones previas, han determinado que un subconjunto de las Lógicas para la Descripción pueden ser traducidas efectivamente a un conjunto de la programación en lógica. Proponemos un método para lidiar con definiciones de ontologías inconsistentes en la Web Semántica. Nuestra propuesta involucra corresponder ontologías expresadas en Lógicas para la Descripción en programas lógicos rebatibles. Esto es, dada una ontología Σ_{OWL} expresada en el lenguaje OWL-DL, es posible construir una ontología Σ_{DL} equivalente expresada en las Lógicas para la Descripción. En el caso en que Σ_{DL} satisfaga ciertas restricciones, ésta puede ser expresada como un programa DeLP $\mathcal{P}_{\Sigma}$. Por lo tanto, dada una consulta ϕ acerca de la pertenencia de una instancia a a un cierto concepto C expresada con respecto a Σ_{OWL}, se realizará un análisis dialéctico para determinar todas las razones a favor y en contra de la plausibilidad de la afirmación $C(a)$. Además, como parte de este acercamiento presentaremos una aplicación a la integración de ontologías. Los resultados presentados en este capítulo están parcialmente basados en los trabajos (Gómez et al., 2006*a*), (Gómez et al., 2007*b*), (Gómez et al., 2008*a*) y (Gómez et al., 2008*c*).

4.1. Introducción y motivaciones

4.1.1. Definición del problema

La *Web Semántica* (Berners-Lee et al., 2001) (véase el capítulo 2 en la página 19) es una visión futura de la Web en la cual la información tiene un significado exacto; permitiendo así que los agentes inteligentes entiendan y razonen en base a la información allí presente. En la Web Semántica se propone resolver el problema de la asignación de semántica a los recursos web por medio de *definiciones de ontologías*, que son utilizadas para dar una descripción precisa de los datos que se encuentran en un recurso web. Un recurso es simplemente un repositorio de datos.

En el estándar propuesto por el *Consorcio de la World Wide Web* (W3C)[1], las definiciones de ontologías se realizan en el lenguaje OWL (por *Ontology Web Language*) (McGuiness and van Harmelen, 2004), cuya semántica está basada en las *Lógicas para la Descripción* (DL) (Baader et al., 2003) (véase el Capítulo 2). Las ontologías expresadas en las DL son definidas en términos de un conjunto de *axiomas terminológicos* (llamado *Tbox*), que contienen información de clases, y un conjunto de *axiomas asercionales* (llamado *Abox*), los cuales contienen información de individuos que pertenecen a las clases definidas en la Tbox.

En el marco de las DL tradicionales, existen razonadores (*e.g.*, FACT (Horrocks, 1998) y RACER (Haarslev and Möller, 2001)) que resuelven en forma eficiente[2] los distintos tipos de problemas de inferencias relevantes a las ontologías. Los principales tipos de inferencias son: la determinación de la pertenencia de un individuo a una cierta clase; conocer si una terminología es consistente, y determinar si una clase es subclase de otra.

Pero, si bien las ontologías *consistentes* expresadas en DL pueden ser manipuladas por razonadores DL, debido a que las ontologías son entes complejos, creados por uno o varios ingenieros de conocimiento, es posible que surjan *inconsistencias* por diversos motivos (veáse la Sección 2.4 en la página 47). En el marco de la representación de conocimiento, una inconsistencia se halla caracterizada por una contradicción lógica. El problema que surge entonces en el marco de trabajo tradicional es que los razonadores DL estándares no son capaces de lidiar con estas ontologías inconsistentes. Típicamente, dada una terminología inconsistente, RACER indicará que ciertas clases deben ser vacías. En el caso en que el ingeniero de conocimiento hubiera manifestado que ciertos individuos pertenecen a esas clases conflictivas, el razonador detendrá su operación indicando que la caja asercional es inconsistente.

[1] www.w3c.org

[2] El análisis de complejidad temporal de peor caso para el problema de la satisfacibilidad para bases de conocimiento expresadas en OWL-DL y OWL-Lite (los que corresponden a las DLs $\mathcal{SHOIN}(D)$ y $\mathcal{SHIF}(D)$, respectivamente) indica que tienen complejidad exponencial (Horrocks and Patel-Schneider, 2004). Sin embargo, en la literatura se sostiene que RACER (Haarslev and Möller, 2001) brinda una implementación altamente optimizada.

En la literatura de la Inteligencia Artificial se consideran dos métodos para lidiar con las inconsistencias (Subrahmanian and Amgoud, 2007): El primer acercamiento consiste en revisar la base de conocimiento y restaurar su consistencia. El segundo acercamiento acepta la inconsistencia y lidia con ella. De acuerdo a (Subrahmanian and Amgoud, 2007), el primer acercamiento fue iniciado por Rescher y Manor (Rescher and Manor, 1970) y propone descartar algunas fórmulas de la base de conocimiento para obtener una de las subbases consistentes de la base de conocimiento original. Entonces, las conclusiones plausibles pueden ser obtenidas utilizando inferencia clásica a partir de dichas subbases. Una manera posible de trabajo consiste en *reparar* la ontología inconsistente para que los razonadores DL puedan trabajar con ellas. Por ejemplo, un acercamiento posible consiste de identificar las subontologías minimalmente insatisfacibles o calcular las subontologías maximalmente satisfacibles (Lam et al., 2006). Sin embargo, ciertos autores (*e.g.*, (Huang et al., 2004, 2005)) argumentan que, en el contexto de las ontologías DL, este enfoque no es viable por varios motivos: primero, los mecanismos de reparación de ontologías estudiados hasta el momento (*e.g.*, (Schlobach and Cornet, 2003)) no escalan al tamaño de la web; segundo, si nosotros no somos los autores de una cierta ontología, no tendremos autoridad suficiente para garantizar que el conocimiento que se elimina de la misma no es crítico a la aplicación en cuestión; tercero, estudios recientes (*e.g.*, (Flouris, 2006)) indican que no es trivial adaptar las Teorías de Revisión de Creencias estándar (*e.g.*, el modelo AGM (Alchourron et al., 1985)) a las Lógicas para la Descripción que subyacen a los lenguajes de la Web Semántica (como OWL (McGuiness and van Harmelen, 2004) y véase Sección 2.2.5 en la página 25).

Como mencionábamos más arriba, otro enfoque consiste en *aceptar* las inconsistencias y lidiar con ellas por medio de un formalismo de razonamiento no monótono. Investigaciones recientes (Grosof et al., 2003; Motik et al., 2005) han mostrado cómo es posible expresar un subconjunto de las ontologías DL en la Programación en Lógica. Sin embargo, tales acercamientos sólamente consideran el razonamiento con ontologías *consistentes*. En este respecto, la *Programación en Lógica Rebatible* (DeLP) (García and Simari, 2004) provee un lenguaje basado en la Programación en Lógica para la representación de conocimiento y el razonamiento que utiliza la *argumentación rebatible* (Chesñevar, Maguitman and Loui, 2000; Prakken and Vreeswijk, 2002; Simari and Loui, 1992) para decidir entre conclusiones *contradictorias* obtenidas a partir de una base de conocimiento (expresada como un programa lógico rebatible) potencialmente *inconsistente* a través de un *análisis dialéctico* (veáse el Capítulo 3).

En base a estas observaciones, en este capítulo exploramos la utilización de la argumentación rebatible para obtener respuestas significativas al problema de la determinación de la pertenencia de individuos a una clase en presencia de ontologías inconsistentes en el marco de la Web Semántica. Los lenguajes de representación de conocimiento en la Web Semántica OWL-DL y OWL-Lite se corresponden con las Lógicas para la Descripción

$\mathcal{SHOIN}(D)$ y $\mathcal{SHIF}(D)$, respectivamente. Dada una ontología expresada en OWL-DL (resp. OWL-Lite) es posible obtener una ontología equivalente expresada en $\mathcal{SHOIN}(D)$ (resp. $\mathcal{SHIF}(D)$). Así, en el caso en que la ontología en cuestión satisfaga ciertas restricciones particulares (véase la Sección 4.2), la propuesta de trabajo consistirá en transformar una ontología expresada en una Lógica para la Descripción en un Programa Lógico Rebatible. En el caso en que la ontología sea inconsistente, se podrá determinar la pertenencia de un cierto individuo a una clase particular utilizando un análisis dialéctico que considerará las razones a favor y en contra de la pertenencia de dicho individuo a esa clase. De este acercamiento, también se desprenderán las soluciones a otros problemas de inferencia típicos del área de las Lógicas para la Descripción pero adaptados a la propuesta de razonamiento argumentativo. A continuación damos una explicación de la solución general al problema descripto.

4.1.2. Esquema general de la solución

Nuestro acercamiento se basará en brindar mecanismos de razonamiento alternativos para las Lógicas para la Descripción (DL) basados en la argumentación rebatible. Para ello, definiremos primeramente una forma de representar el conocimiento contenido en las ontologías DL en el marco de un lenguaje de argumentación rebatible. Este lenguaje consistirá de la Programación en Lógica Rebatible (DeLP). Investigaciones previas sugieren un método para expresar un subconjunto de las DL en el lenguaje de la Programación en Lógica (Grosof et al., 2003; Motik et al., 2005).

La *Web Semántica* (Berners-Lee et al., 2001) es una visión futura de la Web donde la información almacenada tiene un significado exacto, de esta manera permitiendo que agentes inteligentes razonen sobre la base de tal información. La asignación de semántica a los recursos web es realizada por medio de *definiciones de ontologías*. El término *ontología* se refieren a la especificación de una conceptualización; es decir, a la descripción de los conceptos y relaciones que pueden existir para un agente o una comunidad de agentes (Gruber, 1993) (para más detalles, ver el Capítulo 2).

Las definiciones de ontologías se escriben en un *lenguaje de ontologías* como OWL (McGuiness and van Harmelen, 2004), cuyo subconjunto conocido como OWL-DL está basado en las *Lógicas para la Descripción* (DL). A pesar de que las ontologías expresadas en DL pueden ser procesadas por razonadores estándar (como RACER (Haarslev and Möller, 2001)), tales razonadores DL son incapaces de poder lidiar con definiciones de ontologías *inconsistentes*. En particular, cuando a un razonador DL se le presenta una ontología inconsistente no será capaz de obtener información útil a partir de ella. [3]

Sin embargo, investigaciones previas (Grosof et al., 2003) han determinado que un

[3]Un contraejemplo notable es *Pellet* (Parsia and Sirin, 2004), que brinda cierto soporte para manejar ontologías inconsistentes.

subconjunto de las DL puede ser efectivamente traducido a un subconjunto de la Lógica de Horn. En particular, como vimos en el Capítulo 3, la *Programación en Lógica Rebatible* (DeLP) es un marco argumentativo que es capaz de lidiar con bases de conocimiento posiblemente inconsistentes codificadas como un conjunto de cláusulas de Horn (García and Simari, 2004). Cuando se le presenta a DeLP una consulta Q con respecto a una base de conocimiento $\mathcal{P}$, DeLP realiza un proceso *dialéctico* en el cual todos los *argumentos* a favor y en contra de Q son considerados. En tal proceso, los argumentos considerados en última instancia como *vencedores* serán considerados *garantizados*.

En este capítulo, proponemos un método para lidiar con definiciones de ontologías inconsistentes en la Web Semántica. Nuestro acercamiento involucra hacer corresponder ontologías expresadas en DL con un programa DeLP. Es decir, dada una ontología OWL-DL Σ_{Owl}, se puede obtener una ontología DL Σ_{DL} equivalente. Bajo la hipótesis de que Σ_{DL} satisface ciertas restricciones que serán presentadas oportunamente, Σ_{DL} puede ser traducida a un programa DeLP Σ_{DeLP}. Luego, en el caso en que la ontología Σ_{DL} sea inconsistente, dada una consulta Q realizada con respecto a la ontología Σ_{DL}, la misma podrá ser resuelta como una consulta Q con respecto al programa DeLP Σ_{DeLP}. Así, se realizará un análisis dialéctico para determinar si Q está garantizado con respecto Σ_{DeLP}.

La visión general de la solución propuesta se aprecia en la Figura 4.1. Dadas una ontología (T, A) expresada en DL y una consulta ϕ con respecto a dicha ontología, si la consulta fuera resuelta con un razonador estándar DL, obtendríamos una respuesta $\mathcal{R}_{DL}$. Sin embargo, veremos cómo representar la ontología DL (T, A) como un programa DeLP (Π, Δ). Tal cambio de representación se realizará utilizando una función de traducción $\mathfrak{T}$ que toma una ontología DL y la hace corresponder con un programa DeLP. Análogamente, la consulta ϕ será apropiadamente re-expresada como una consulta ϕ'. Al realizar la consulta ϕ' respecto del programa DeLP (Π, Δ), se obtendrá una respuesta $\mathcal{R}_{DeLP}$. Finalmente, en el Capítulo 5 estudiaremos qué relación existe entre las respuestas devueltas por un razonador DL y por el motor de razonamiento DeLP para distintos tipos de ontologías.

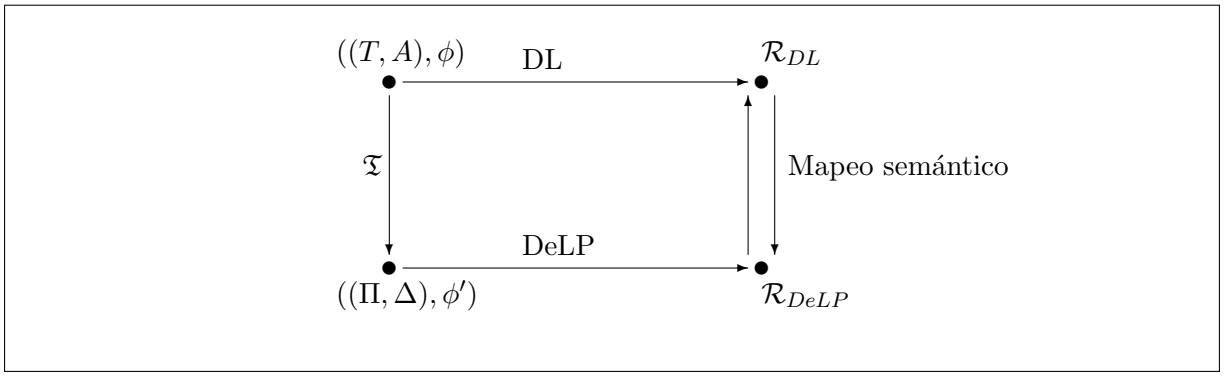


Figura 4.1: Visión general de la solución propuesta

El resto del capítulo está estructurado de la siguiente manera: En la Sección 4.2, estudiamos las relaciones subyacentes entre las Lógicas para la Descripción, la Lógica de Primer Orden y la Programación en Lógica Rebatible, que permitirán expresar los distintos tipos de construcciones DL como una o más reglas DeLP. En la Sección 4.3, presentamos una adaptación de la correspondencia de Grosof *et al.* para mapear ontologías DL en programas DeLP. Seguidamente, en la Sección 4.4, definimos a las δ-ontologías, que son ontologías cuya semántica está basada en la argumentación rebatible. En la Sección 4.5, mostramos cómo ciertas tareas de razonamiento para las DL tradicionales pueden ser resueltas en el marco de las δ-ontologías. En la Sección 4.6 estudiamos la aplicación del acercamiento propuesto a la integración de ontologías. En la Sección 4.7 discutimos estrategias para la obtención de subconjuntos maximales consistentes a partir de ontologías DL inconsistentes. En la Sección 4.8 presentamos los trabajos relacionados de la literatura. Finalmente, la Sección 4.9 concluye el capítulo.

4.2. Expresando ontologías de las Lógicas para la Descripción en la Programación en Lógica Rebatible

Como se explicó anteriormente, las ontologías expresadas en las DL pueden ser inconsistentes. Notablemente, razonadores DL como RACER (Haarslev and Möller, 2001) son incapaces de lidiar con tales situaciones. Al ser presentado ante tales ontologías inconsistentes, RACER emitirá un mensaje de error y abortará cualquier procesamiento posterior.

Por otro lado, investigaciones recientes proponen el procesamiento de las ontologías expresadas en las DL en el marco de la programación en lógica de tal manera de aprovechar las múltiples implementaciones del lenguaje Prolog existentes (Lloyd, 1987; Sterling and Shapiro, 1994; Baral, 2003). En este respecto, Grosof *et al.* (Grosof et al., 2003)[4] han identificado un subconjunto de los lenguajes de DL que pueden ser efectivamente mapeados en una lógica de cláusulas de Horn, por lo que basaremos nuestro trabajo en tales investigaciones adaptándolas al marco de DeLP. Por lo tanto, proponemos traducir una ontología DL $\Sigma = (T, A)$ en un programa DeLP $\mathcal{P} = (\Pi, \Delta)$ por medio de una correspondencia $\mathfrak{T}$ tal que $\mathcal{P} = \mathfrak{T}(\Sigma)$. Intuitivamente, el conjunto (de hechos) Π en $\mathcal{P}$ se corresponderá con el conjunto de aserciones A en Σ junto con ciertas sentencias de la caja terminológica T. El conjunto Δ se corresponderá con un subconjunto (posiblemente *inconsistente*) de la caja terminológica T de Σ. En el resto de esta sección, explicaremos cómo obtener la traducción de bases de conocimiento expresadas en DL hacia programas DeLP. Además, en algunos casos, las reglas rebatibles de la forma "$H \prec B_1, \ldots, B_n$"

[4]Este trabajo muy probablemente es la versión en forma de artículo que reporta los resultados hallados por Volz en el contexto de su trabajo de tesis de doctorado (Volz, 2004). Otros trabajos en los que participan algunos de los autores de (Grosof et al., 2003) y también reportan resultados en este respecto son (Motik et al., 2003, 2005).

serán escritas por claridad como "$H \prec B_1 \wedge \ldots \wedge B_n$", mientras que las reglas estrictas de la forma "$H \leftarrow B_1, \ldots, B_n$" serán escritas como $H \leftarrow B_1 \wedge \ldots \wedge B_n$"; sin embargo, cuando no exista confusión seguiremos utilizando la notación clásica de DeLP para expresar tanto reglas estrictas como rebatibles.

4.2.1. Representación de sentencias

Para mantener compatibilidad hacia atrás con lenguajes de representación de conocimiento en la Web Semántica, parte de los constructores de OWL DL están basados en el lenguaje RDF Schema (RDFS). RDFS provee un subconjunto de las sentencias DL (sentencias de subclase, subpropiedad, rango y dominio), los cuales en un marco DL son llamados axiomas *Tbox*, y relaciones de instancia de clases (tipo) y de instancias de relaciones entre instancias (las cuales en un marco DL son llamadas axiomas *Abox*). Un axioma de inclusión DL se corresponde con una implicación de Lógica de Primer Orden (FOL)[5].

Entonces, los axiomas de clase y propiedad se mapean en DeLP como:

Definición 4.2.1 (Representación de relaciones de subclase y subpropiedad) *Sean C y D nombres de clases, X, Y nombres de variables y P, Q nombres de propiedades. La clase C es subclase de la clase D, notado en DL como "$C \sqsubseteq D$", en el caso en que la relación no tenga excepciones se mapea a DeLP como "$D(X) \leftarrow C(X)$" y en el caso en que tenga excepciones se mapea a DeLP como "$D(X) \prec C(X)$". La propiedad Q es una subpropiedad de la propiedad P, notado como "$Q \sqsubseteq P$", mapea a "$P(X, Y) \leftarrow Q(X, Y)$" si no tiene excepciones, y a "$P(X, Y) \prec Q(X, Y)$" si puede tener excepciones.*

Las sentencias de rango y dominio sirven para establecer el tipo de los atributos y se mapean como se define a continuación:

Definición 4.2.2 (Representación de dominio y rango de propiedades) *Sean C un nombre de clase, P un nombre de propiedad y X, Y nombres de variable. El rango de la propiedad P es la clase C, notado como $\top \sqsubseteq \forall P.C$, se mapea a "$C(Y) \leftarrow P(X, Y)$" (caso sin excepciones) o a "$C(Y) \prec P(X, Y)$"(caso con posibles excepciones). El dominio de la propiedad P está contenido en la clase C, notado como "$\top \sqsubseteq \forall P.C$", se mapea a "$C(Y) \leftarrow P(Y, X)$" (caso sin excepciones) o a "$C(Y) \prec P(Y, X)$" (caso con excepciones).*

Definición 4.2.3 (Representación de instancias de clase y de propiedades) *Sean C un nombre de clase, P un nombre de propiedad, a, b nombres de individuos. Las relaciones de instancia de clase y de propiedades, las cuales corresponden a los axiomas DL*

[5]Por ejemplo, el axioma DL $C \sqsubseteq D$ se corresponde con la fórmula FOL $(\forall x)(C(x) \rightarrow D(x))$. Para más detalles, véase la Sección 2.3.3.

*de la forma "$a : C$" y "$\langle a, b\rangle : P$" respectivamente (*i.e., *axiomas Abox), son equivalentes a los hechos DeLP de la forma "$C(a)$" y "$P(a, b)$", donde a y b son constantes.*

Ejemplo 4.2.1 (Representación de la relación de subclase) *Consideremos la sentencia DL:*

$$Cirujano \sqsubseteq Persona$$

que representa la relación de subclase entre las clases "Cirujano" y "Persona". Esta relación se representará como regla estricta de la forma:

$$persona(X) \leftarrow cirujano(X)$$

y como regla rebatible de la forma:

$$persona(X) \prec cirujano(X)$$

Ejemplo 4.2.2 (Representación de la subrelaciones) *Consideremos una ontología donde se expresa que las relaciones "pintar" y "restaurar" son subrelaciones de la relación "modificar". En DL esta idea se expresa con las sentencias:*

$$pintar \sqsubseteq modificar$$
$$restaurar \sqsubseteq modificar.$$

Estas sentencias expresadas como reglas estrictas y rebatibles respectivamente se escriben como:

$$modificar(X, Y) \leftarrow pintar(X, Y)$$
$$modificar(X, Y) \prec restaurar(X, Y).$$

Ejemplo 4.2.3 (Representación de atributos) *El dominio de la propiedad "es_mascota_de" es de tipo "Mascota" y el rango es de tipo "Persona", se especifica en DL con las sentencias respectivas:*

$$\top \sqsubseteq \forall es\_mascota\_de.Mascota$$
$$\top \sqsubseteq \forall es\_mascota\_de^{-}.Persona$$

se representarán con las reglas rebatibles:

$$mascota(X) \prec es\_mascota\_de(X, Y)$$
$$persona(Y) \prec es\_mascota\_de(X, Y)$$

Los axiomas de equivalencia de clase y propiedad pueden ser reemplazados por un par simétrico de axiomas de inclusión; luego, pueden ser mapeados a un par simétrico de reglas DeLP como se muestra a continuación:

Definición 4.2.4 (Representación de equivalencias de clase y propiedad) *Sean C, D nombres de clases, P, Q nombres de propiedades, X, Y nombres de variables. La clase C es equivalente a la clase D, notado como $C \equiv D$, mapea al conjunto de reglas DeLP (en el caso de inclusión estricta):*

$$\left\{ \begin{array}{l} C(X) \leftarrow D(X) \\ D(X) \leftarrow C(X) \end{array} \right\},$$

o (en el caso de inclusión con excepciones):

$$\left\{ \begin{array}{l} C(X) \prec D(X) \\ D(X) \prec C(X) \end{array} \right\}$$

La propiedad P es equivalente a la propiedad Q, notado como $P \equiv Q$, mapea al conjunto de reglas DeLP:

$$\left\{ \begin{array}{l} P(X,Y) \leftarrow Q(X,Y) \\ Q(X,Y) \leftarrow P(X,Y) \end{array} \right\}$$

o

$$\left\{ \begin{array}{l} P(X,Y) \prec Q(X,Y) \\ Q(X,Y) \prec P(X,Y) \end{array} \right\}$$

Los axiomas de inversos de propiedades y de transitividad pueden ser traducidos también:

Definición 4.2.5 (Representación de axiomas inversos de propiedad y transitividad) *Sean P, Q nombres de propiedades, X, Y nombres de variables. La propiedad P es la inversa de la propiedad Q, notado como "$P \equiv Q^-$", mapea al conjunto de reglas DeLP en el caso estricto:*

$$\left\{ \; Q(Y,X) \leftarrow P(X,Y); P(X,Y) \leftarrow Q(Y,X) \; \right\}$$

y en el caso rebatible:

$$\left\{ \; Q(Y,X) \prec P(X,Y); P(X,Y) \prec Q(Y,X) \; \right\}.$$

La propiedad P es transitiva, notado como "$P^+ \sqsubseteq P$", se mapea a

$$P(X,Z) \leftarrow P(X,Y), P(Y,Z)$$

y en el caso rebatible:

$$P(X,Z) \prec P(X,Y), P(Y,Z)$$

Ejemplo 4.2.4 (Representación de equivalencia de clases) *Si queremos representar que la clase de las personas es siempre equivalente a la clase de los humanos, lo haremos con la sentencia DL:*

$$Human \equiv Person.$$

Esta sentencia es representada como dos sentencias estrictas en DeLP:

$$human(X) \leftarrow person(X)$$
$$person(X) \leftarrow human(X)$$

Ejemplo 4.2.5 (Representación de equivalencia de relaciones) *Evaluemos el caso en que las relaciones "asegurar" y "proteger" son usualmente equivalentes, representado en DL por:*

$$insure \equiv protect;$$

estas caracterización será representada en DeLP con dos reglas rebatibles:

$$insure(X, Y) \prec protect(X, Y)$$
$$protect(X, Y) \prec insure(X, Y)$$

Ejemplo 4.2.6 (Representación de propiedades inversas) *Si consideramos que siempre la relación "presta" es la inversa "pide", representado en DL con:*

$$presta \equiv pide^{-};$$

esta sentencia será representada en DeLP con las reglas estrictas:

$$presta(X, Y) \leftarrow pide(Y, X)$$
$$pide(X, Y) \leftarrow presta(Y, X)$$

Ejemplo 4.2.7 (Representación de propiedades transitivas) *Si modelamos a la relación de contagio de enfermedades como transitiva, tendremos que:*

$$contagia \sqsubseteq contagia^{+}.$$

Si suponemos que la transitividad del contagio puede tener excepciones, entonces ésta será modelada con reglas rebatibles de la forma:

$$contagia(X, Z) \prec contagia(X, Y), contagia(Y, Z).$$

4.2.2. Representación de constructores de clase

En la sección previa, mostramos cómo los axiomas DL se corresponden con reglas DeLP, y cómo éstas pueden ser usadas para realizar afirmaciones sobre clases y propiedades. En las DLs, las expresiones de clases que aparecen en tales axiomas no son necesariamente atómicas ya que pueden ser expresiones complejas formadas a partir de clases y propiedades atómicas utilizando una variedad de constructores. A continuación, mostraremos cómo estas expresiones DL se corresponden con expresiones en el cuerpo de reglas DeLP. Usaremos C, D para denotar clases, y P, Q para denotar propiedades.

Conjunciones

Una clase DL puede ser formada por medio de la conjunción de clases existentes (notado como "$C \sqcap D$"), lo cual se corresponde con una conjunción de predicados unarios. La conjunción puede ser directamente expresada en el cuerpo de una regla DeLP.

Definición 4.2.6 (Representación de conjunciones) *Sean C_1, C_2, D nombres de clases y X un nombre de variable. Cuando una conjunción ocurre en el miembro izquierdo de un axioma de sublase (como en "$C_1 \sqcap C_2 \sqsubseteq D$"), se convierte en una conjunción en el cuerpo de la regla correspondiente. Entonces si consideramos a la sentencia sin excepciones, será de la forma:*

$$D(X) \leftarrow C_1(X) \wedge C_2(X).$$

En el caso en que la sentencia pueda contener excepciones, será de la forma:

$$D(X) \prec C_1(X) \wedge C_2(X).$$

Cuando una conjunción ocurre en el miembro derecho de un axioma de subclase (como en "$C \sqsubseteq D_1 \sqcap D_2$"), se convierte en dos reglas DeLP (caso sin excepciones):

$$\left\{ \begin{array}{l} D_1(X) \leftarrow C(X); \\ D_2(X) \leftarrow C(X) \end{array} \right\}$$

o, (caso con excepciones):

$$\left\{ \begin{array}{l} D_1(X) \prec C(X); \\ D_2(X) \prec C(X) \end{array} \right\}.$$

Nótese que cuando una conjunción ocurre en el miembro derecho de un axioma de subclase, se convierte en una conjunción en la cabeza de la regla: "$D_1(X) \wedge D_2(X) \leftarrow C(X)$"; ésta puede ser transformada, a través de una transformación de Lloyd-Topor (Lloyd and Topor, 1984), en dos reglas como se mostraron anteriormente.

Ejemplo 4.2.8 (Representación de conjunciones) *Consideremos la sentencia DL que expresa que las gallinas asustadas usualmente pueden volar:*

$$\mathit{chicken} \sqcap \mathit{scared} \sqsubseteq \mathit{flies}.$$

Ésta es expresada en DeLP con la regla rebatible:

$$\mathit{flies}(X) \prec \mathit{chicken}(X), \mathit{scared}(X).$$

Ahora tengamos en cuenta la sentencia DL que expresa que todos los elefantes son grises y pesados:

$$\mathit{elephant} \sqsubseteq \mathit{grey} \sqcap \mathit{heavy}.$$

Este conocimiento es expresado en DeLP con las reglas estrictas:

$$\mathit{grey}(X) \leftarrow \mathit{elephant}(X)$$
$$\mathit{heavy}(X) \leftarrow \mathit{elephant}(X).$$

Disyunciones

Una clase DL puede ser formada a partir de la disyunción de clases existentes, lo cual se corresponde con una disyunción de predicados unarios. La siguiente definición considera el caso en que la disyunción de clases aparece en un axioma de subclase.

Definición 4.2.7 (Representación de disyunciones) *Sean C_1, C_2, D nombres de clases y X un nombre de variable. En el caso estricto, la sentencia de inclusión "$C_1 \sqcup C_2 \sqsubseteq D$" se convierte en dos reglas estrictas DeLP:*

$$\left\{ \begin{array}{l} D(X) \leftarrow C_1(X); \\ D(X) \leftarrow C_2(X) \end{array} \right\}$$

En el caso rebatible, se convierte a:

$$\left\{ \begin{array}{l} D(X) \prec C_1(X); \\ D(X) \prec C_2(X) \end{array} \right\}$$

Nótese que cuando una disyunción ocurre en el miembro izquierdo de un axioma de subclase, se convierte en una disyunción en el cuerpo de la regla "$D(X) \leftarrow C_1(X) \vee C_2(X)$"; ésta es transformada a través de una transformación de Lloyd-Topor (Lloyd and Topor, 1984) en las reglas presentadas previamente. Nótese también que cuando una disyunción ocurre en el miembro derecho de un axioma de subclase, ésta se convierte en una disyunción en el cuerpo de la cabeza de la regla correspondiente; esto no puede ser representado en DeLP ya que su lenguaje no permite reglas con disyunciones en la cabeza.

Ejemplo 4.2.9 (Representación de disyunciones) *Consideremos el siguiente axioma DL que expresa que siempre el conjunto de los deportistas acuáticos incluye al de los nadadores y al de los buzos:*

$$Nadador \sqcup Buzo \sqsubseteq Deportista\_acuatico$$

Esto es representado con las reglas estrictas DeLP:

$$\left\{ \begin{array}{l} deportista\_acuatico(X) \leftarrow nadador(X) \\ deportista\_acuatico(X) \leftarrow buzo(X) \end{array} \right\}.$$

Restricciones universales

En una DL, el cuantificador universal sólo puede ser usado en restricciones (expresiones de la forma $\forall P.C$, donde P debe ser una propiedad simple primitiva pero C puede ser una expresión compuesta).

Definición 4.2.8 (Representación de restricciones universales) *Sean C, D expresiones de clases, P un nombre de propiedad, X, Y nombres de variables. Una restricción universal en el miembro derecho de un axioma de subclase (*i.e., *$C \sqsubseteq \forall P.D$), en el caso estricto, es expresada en DeLP con una regla estricta:*

$$D(Y) \leftarrow C(X), P(X,Y)$$

y, en el caso tentativo, como una regla rebatible:

$$D(Y) \prec C(X), P(X,Y).$$

Ejemplo 4.2.10 (Representación de restricciones universales) *Por ejemplo, si queremos escribir una ontología sobre DeLP, el siguiente axioma DL:*

$$Undefeated \sqsubseteq \forall is\_attacked\_by.Defeated$$

especifica que un argumento es no derrotado si todo argumento atacante se halla derrotado. Esta regla sería codificada por la regla estricta:

$$defeated(Y) \leftarrow undefeated(X), is\_attacked\_by(X,Y).$$

Restricciones existenciales

En una DL, el cuantificador existencial es usado en una expresión de la forma $\exists P.C$, donde P debe ser una propiedad simple primitiva, pero C puede ser una expresión compuesta. Cuando una restricción existencial ocurre en el miembro izquierdo de un axioma de subclase, ésta es expresada como una conjunción en el cuerpo de una regla DeLP.

Definición 4.2.9 (Representación de restricciones existenciales) *Sea P un nombre de propiedad, C una expresión de clase, X, Y nombres de variables. El axioma de subclase $\exists P.C \sqsubseteq D$ sin excepciones es expresado en DeLP como "$D(X) \leftarrow P(X,Y) \wedge C(Y)$", y cuando puede tener excepciones es expresado como "$D(X) \prec P(X,Y) \wedge C(Y)$".*

Note que cuando una restricción existencial ocurre en el miembro derecho de un axioma de subclase, ésta debería ser expresada como una conjunción en la cabeza de la regla correspondiente, con una variable cuantificada existencialmente. Esta situación no puede ser representada en DeLP, ya que no puede ser representada en el marco de la Programación en Lógica.

Ejemplo 4.2.11 (Representación de restricciones existenciales) *Consideremos el axioma DL:*

$$\exists owns.Dog \sqsubseteq PetOwner$$

que expresa que todos los individuos que son dueños de un perro están contenidos en el conjunto de los individuos que son dueños de mascotas. Si consideramos a este axioma como tentativo, entonces el mismo será expresado con la regla rebatible:

$$petOwner(X) \prec owns(X,Y), dog(Y).$$

4.3. Una correspondencia de DL a DeLP

En (Grosof et al., 2003), Grosof *et al.* presentan una correspondencia desde una Lógica para la Descripción hacia un subconjunto de la Programación en Lógica. En consecuencia, dicha correspondencia toma como argumento una ontología expresada en DL y devuelve un programa lógico. Para que dicha correspondencia pueda ser efectivamente calculada, la ontología DL sobre la cual se aplica la correspondencia debe satisfacer las restricciones que se comentaron en la sección anterior. Volz (Volz, 2004) muestra que bajo las restricciones impuestas es posible expresar entre el 77 y el 87 % de las ontologías analizadas. Dentro de las ontologías analizadas, es posible expresar entre el 90 y el 99 % de los axiomas individuales en cada una de ellas.

En esta sección, presentaremos una adaptación de tal mapeo para traducir una base de conocimiento a un programa DeLP. Un acercamiento posible a este problema fue abordado en (Gómez et al., 2006*a*, 2007*b*). En dicho acercamiento, todas las sentencias de la Tbox original son expresadas como reglas rebatibles en el programa DeLP resultante de la transformación, mientras que todas las sentencias de aserción de la Abox son interpretadas como hechos en el programa DeLP resultante de la transformación. Esto se puede interpretar como que todas las inclusiones de conjuntos expresadas por la Tbox son consideradas como reglas *default*.

En ciertas aplicaciones, es interesante que ciertas sentencias de la Tbox sean interpretadas como reglas estrictas y otras como reglas rebatibles. De hecho, al considerar la información contenida en la Abox, ciertos axiomas de inclusión de la Tbox quedarán invalidados. Esta invalidación se produce al detectarse *excepciones* en los axiomas. Por ejemplo, supongamos que tenemos una ontología que contiene un axioma Tbox "$C \sqsubseteq D$" y descubrimos que en la Abox tenemos dos aserciones de la forma "$a : C$" y "$\neg a : D$". Por lo tanto, desde el punto de vista del razonamiento tradicional con ontologías, el axioma de inclusión queda invalidado porque existe un individuo "a" que pertenece al concepto "C" y no pertenece al concepto "D". Sin embargo, la misma Abox puede contener como parte de su información a las aserciones "$b : C$", "$b : D$", "$e : C$" y "$e : D$" las cuales satisfacen a la regla en cuestión. En conclusión, ya que el axioma "$C \sqsubseteq D$" es *usualmente* satisfecho pero sin embargo en algunos casos *excepcionales* no lo es, el mismo se torna *rebatible*.

Formalmente, dada una ontología $\Sigma = (T, A)$, nos interesará particionar al conjunto T en un conjunto de sentencias *estrictas* T_S y otro conjunto de sentencias *rebatibles* T_D, tal que $T_S \cup T_D = T$ y $T_S \cap T_D = \emptyset$. Luego, de poder construir una función $\mathfrak{T}_\Pi$ que dada una sentencia DL la correspondiera con una regla estricta en un programa DeLP, y de poder construir una función $\mathfrak{T}_\Delta$ que dada una sentencia DL la correspondiera con una regla rebatible en un programa DeLP, el programa DeLP obtenido aplicando la transformación sería de la forma: $\mathcal{P} = (\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$. En vista de esta observación, planteamos

un nuevo acercamiento donde parte de la Tbox será expresada como un conjunto de reglas estrictas y otra parte como reglas rebatibles mientras que la información de la Abox será expresada como un conjunto de hechos. Esta variante fue presentada en (Gómez et al., 2008*a*).

En el resto de esta sección presentamos este enfoque. Primeramente, establecemos las definiciones preliminares para completar la función de traducción de DL a DeLP. En segundo lugar, resolvemos el problema de expresar axiomas terminológicos como reglas rebatibles. Finalmente, abordamos el problema de representar axiomas terminológicos y asercionales como reglas estrictas y hechos respectivamente.

4.3.1. Preliminares

En las secciones previas, hemos identificado todos los casos en relación a la traducción de sentencias y axiomas DL hacia DeLP. Utilizando tales definiciones como guía, cada axioma DL es mapeado hacia una o más reglas DeLP. Vimos que ciertos constructores de las DL, como conjunción y restricción universal, pueden ser mapeados a las cabezas de reglas cuando ocurren en el miembro derecho de un axioma de inclusión. En cambio, otros constructores, como la conjunción, disyunción y restricción existencial, pueden ser mapeados a los cuerpos de reglas cada vez que ocurren en el miembro izquierdo de un axioma de inclusión. De acuerdo a estas observaciones, Grosof (Grosof et al., 2003) definen dos lenguajes llamados $\mathcal{L}_h$ y $\mathcal{L}_b$, que corresponden a los conjuntos de sentencias que pueden ser mapeados a la cabezas y cuerpos de reglas de la programación en lógica, respectivamente.

Definición 4.3.1 (Lenguaje $\mathcal{L}_h$. $\mathcal{L}_h$-clases (adaptado de (Grosof et al., 2003))) *Sea A un nombre atómico de clase, C y D expresiones de clases, y R una propiedad. En el lenguaje $\mathcal{L}_h$, $C \sqcap D$ es una clase, y $\forall R.C$ es también una clase. Las expresiones de clases en el lenguaje $\mathcal{L}_h$ se llamarán $\mathcal{L}_h$-clases.*

Definición 4.3.2 (Lenguaje $\mathcal{L}_b$. $\mathcal{L}_b$-clases (adaptado de (Grosof et al., 2003))) *Sea A un nombre atómico de clase, C y D expresiones de clases, y R una propiedad. En el lenguaje $\mathcal{L}_b$, $C \sqcap D$, $C \sqcup D$ y $\exists R.C$ son clases. Las expresiones de clases en el lenguaje $\mathcal{L}_b$ se llamarán $\mathcal{L}_b$-clases.*

Debido a que los axiomas de igualdad de la forma $C \equiv D$ se traducen como dos axiomas de inclusión $C \sqsubseteq D$ y $D \sqsubseteq C$, para que se cumplan las restricciones impuestas por las definiciones anteriores, es necesario definir la intersección de ambos lenguajes.

Definición 4.3.3 (Lenguaje $\mathcal{L}_{hb}$. $\mathcal{L}_{hb}$-clases (adaptado de (Grosof et al., 2003))) *Sea A un nombre atómico de clase, C y D expresiones de clases, y R una propiedad. El lenguaje $\mathcal{L}_{hb}$ se define como la intersección de los lenguajes $\mathcal{L}_h$ y $\mathcal{L}_b$. Formalmente,*

Cuadro 4.1: Reglas para la normalización/simplificación de descripciones de clases (Volz, 2004)

$\neg\forall R.C \mapsto \exists R.C$	$\neg\exists R.C \mapsto \forall R.\neg C$	$\leq nR \mapsto \neg \geq (n+1)R$
$\neg(\sqcap_i C_i) \mapsto \sqcup_i \neg C_i$	$\neg(\sqcup_i C_i) \mapsto \sqcap_i(\neg C_i)$	$\neg\top \mapsto \bot$
$\top \sqcap C \mapsto C$	$\bot \sqcap C \mapsto \bot$	$\forall R.\top \mapsto \top$
$\top \sqcup C \mapsto \top$	$\bot \sqcup C \mapsto C$	$\exists R.\bot \mapsto \bot$
$\leq 0R \mapsto \forall R.\bot$	$\geq 1R \mapsto \exists R.\top$	$\geq 0R \mapsto \top$

$$\mathcal{L}_{hb} =_{def} \mathcal{L}_h \cap \mathcal{L}_b.$$

Las expresiones de clases en el lenguaje $\mathcal{L}_{hb}$ se llamarán $\mathcal{L}_{hb}$-clases.

Definición 4.3.4 (Lenguaje de los programas DeLP estrictos) *$\mathcal{L}_{DeLP_\Pi}$ es el lenguaje de los programas DeLP conformado solamente por reglas estrictas $B \leftarrow A_1, \ldots, A_n$ con $(n \geq 1)$ y hechos B (*i.e.*, reglas donde $n = 0$).*

Definición 4.3.5 (Lenguaje de los programas DeLP rebatibles) *$\mathcal{L}_{DeLP_\Delta}$ es el lenguaje de los programas DeLP conformado solamente por reglas rebatibles $B \prec A_1, \ldots, A_n$ con $(n \geq 1)$.*

Definición 4.3.6 (Lenguaje de los programas DeLP) *Definiremos a $\mathcal{L}_{DeLP}$ como el lenguaje de los programas DeLP. Además, $\mathcal{L}_{DeLP}$ se definirá como la unión de los programas DeLP estrictos y rebatibles,* i.e.*: $\mathcal{L}_{DeLP} =_{def} \mathcal{L}_{DeLP_\Pi} \cup \mathcal{L}_{DeLP_\Delta}$.*

En lo que sigue supondremos las ontologías como normalizadas. La normalización consiste en la reducción del número total de combinaciones de constructores que se deben considerar de tal manera de maximizar el número de axiomas que se deben traducir de DL a programación en lógica. Este proceso se basa en tres técnicas y se explican en detalle en la literatura (Volz, 2004). Las dos primeras son referidas como normalización y simplificación de descripciones de clases y axiomas. El tercer tipo es referido como transformación estructural, la cual no es necesariamente requerida, pero permite concentrarse en constructores DL individuales y no en descripciones de clase arbitrariamente complejas.

Para definir la correspondencia entre sentencias DL y reglas DeLP es necesario asumir que las definiciones de ontologías se hallan normalizadas con respecto a la negación. Es decir, las negaciones en expresiones de clases son movidas *hacia adentro* en las expresiones hasta aparecer solamente como negaciones de nombres de conceptos. Esta transformación de axiomas DL se realiza utilizando las reglas de De Morgan y las conocidas equivalencias entre distintas expresiones como restricciones existenciales y de valor y la transformación de restricciones numéricas, las cuales se presentan en el Cuadro 4.1 y la misma está tomada de (Volz, 2004).

A continuación, definiremos una correspondencia de DL a DeLP. Sin perder generalidad, asumiremos que las expresiones de la ontología se hallan normalizadas respecto de la negación (véase la Definición 2.3.1 en la página 32).

4.3.2. Representación de axiomas de inclusión e igualdad como reglas rebatibles

Como comentamos en la introducción de este capítulo, nuestro interés en el acercamiento propuesto en esta tesis es el de razonar en presencia de ontologías inconsistentes. Las ontologías pueden ser representadas como axiomas de inclusión o igualdad perteneciendo a una terminología. Por su parte, los miembros izquierdo y derecho de los axiomas de inclusión o igualdad representarán expresiones de clase. Dichas expresiones de clase pueden corresponder a nombres de clases simples o a expresiones complejas. En la secciones anteriores, vimos cómo casos particulares de tales expresiones pueden ser expresadas en el marco de la programación en lógica; en particular, vimos también las restricciones que deben poseer tales expresiones para poder lograr tal cometido.

Así, si bien los razonadores DL estándar son capaces de lidiar con terminologías consistentes, nosotros estamos interesados en poder representar ontologías inconsistentes y poder razonar con ellas en forma automatizada. En la Sección 2.4 (página 47) presentamos diversos motivos explicando por qué una ontología puede ser inconsistente. En particular, nos interesa poder representar las terminologías inconsistentes en el marco de la Programación en Lógica Rebatible. De esta manera, en nuestro acercamiento al razonamiento con ontologías DL inconsistentes, las tareas de inferencia típicas del marco DL involucrarán realizar un análisis dialéctico para sopesar razones a favor y en contra de una determinada conclusión.

En esta sección, presentamos una función de traducción de terminologías DL como reglas rebatibles. Para ello, se tendrán en cuenta las restricciones comentadas en la Sección 4.2. Finalmente, veremos cómo representar en este contexto las ontologías inconsistentes presentadas en la Sección 2.4 para más adelante presentar la forma en la que se resolverán las tareas de inferencia de las DL pero en el contexto de la argumentación rebatible.

Definición 4.3.7 (Correspondencia $\mathfrak{T}_\Delta$ desde sentencias DL hacia reglas rebatibles DeLP) *Sean A, C, D conceptos, X, Y variables, P, Q relaciones. La correspondencia $\mathfrak{T}_\Delta : 2^{\mathcal{L}_{DL}} \rightarrow 2^{\mathcal{L}_{DeLP_\Delta}}$ se define como se presenta en la Figura 4.2. Además, las reglas de la forma "$(H_1 \wedge H_2) \prec B$" son reescritas como dos reglas "$H_1 \prec B$" y "$H_2 \prec B$", las reglas de la forma "$H_1 \prec H_2 \prec B$" son reescritas como "$H_1 \prec B \wedge H_2$", y las reglas de la forma "$H \prec (B_1 \vee B_2)$" son reescritas como dos reglas "$H \prec B_1$" y "$H_1 \prec B_2$".*

A continuación, presentamos la codificación como reglas rebatibles de los ejemplos

$$
\begin{array}{rcl}
\mathfrak{T}_\Delta(\{C \sqsubseteq D\}) & =_{def} & \left\{ T_h(D,X) \prec T_b(C,X) \right\}, \\
 & & \text{si } C \text{ es una } \mathcal{L}_b\text{-clase y } D \text{ una } \mathcal{L}_h\text{-clase} \\
\mathfrak{T}_\Delta(\{C \equiv D\}) & =_{def} & \mathfrak{T}_\Delta(\{C \sqsubseteq D\}) \cup \mathfrak{T}_\Delta(\{D \sqsubseteq C\}), \text{ si } C \text{ y } D \text{ son } \mathcal{L}_{hb}\text{-clases} \\
\mathfrak{T}_\Delta(\{\top \sqsubseteq \forall P.D\}) & =_{def} & \left\{ T_h(D,Y) \prec P(X,Y) \right\}, \text{ si } D \text{ es una } \mathcal{L}_h\text{-clase} \\
\mathfrak{T}_\Delta(\{\top \sqsubseteq \forall P^-.D\}) & =_{def} & \left\{ T_h(D,X) \prec P(X,Y) \right\}, \text{ si } D \text{ es una } \mathcal{L}_h\text{-clase} \\
\mathfrak{T}_\Delta(\{P \sqsubseteq Q\}) & =_{def} & \left\{ Q(X,Y) \prec P(X,Y) \right\} \\
\mathfrak{T}_\Delta(\{P \equiv Q\}) & =_{def} & \left\{ \begin{array}{l} Q(X,Y) \prec P(X,Y) \\ P(X,Y) \prec Q(X,Y) \end{array} \right\} \\
\mathfrak{T}_\Delta(\{P \equiv Q^-\}) & =_{def} & \left\{ \begin{array}{l} Q(X,Y) \prec P(Y,X) \\ P(Y,X) \prec Q(X,Y) \end{array} \right\} \\
\mathfrak{T}_\Delta(\{P^+ \sqsubseteq P\}) & =_{def} & \left\{ P(X,Z) \prec P(X,Y) \wedge P(Y,Z) \right\} \\
\mathfrak{T}_\Delta(\{s_1,\ldots,s_n\}) & =_{def} & \bigcup_{i=1}^{n} \{\mathfrak{T}_\Delta(\{s_i\})\}, \text{ si } n > 1
\end{array}
$$

donde:

$$
\begin{array}{rcl}
T_h(A,X) & =_{def} & A(X) \\
T_h((C \sqcap D),X) & =_{def} & T_h(C,X) \wedge T_h(D,X) \\
T_h((\forall R.C),X) & =_{def} & T_h(C,Y) \prec R(X,Y) \\
T_b(A,X) & =_{def} & A(X) \\
T_b((C \sqcap D),X) & =_{def} & T_b(C,X) \wedge T_b(D,X) \\
T_b((C \sqcup D),X) & =_{def} & T_b(C,X) \vee T_b(D,X) \\
T_b((\exists R.C),X) & =_{def} & R(X,Y) \wedge T_b(C,Y)
\end{array}
$$

Figura 4.2: Traducción de sentencias de ontologías DL a reglas rebatibles DeLP

presentados en la Sección 2.4 con motivo de mostrar las fuentes de posibles inconsistencias en la Web Semántica cuando el conocimiento se codifica utilizando las Lógicas para la Descripción. Se debe notar que en muchos ejemplos, traduciremos conceptos "C" y "D" como nombres de predicados "c" y "d" para indicar que no son metavariables en las reglas DeLP; es decir, los axiomas de inclusión de la forma "$C \sqsubseteq D$" como "$d(X) \prec c(X)$".

Ejemplo 4.3.1 (Continúa el Ejemplo 2.4.1) *Consideremos nuevamente el Ejemplo 2.4.1 presentado en la página 48. Cuando aplicamos la función de traducción $\mathfrak{T}_\Delta$ a la terminología allí presentada obtenemos el siguiente conjunto $\Delta_{2.4.1}$ de reglas rebatibles:*

$$
\Delta_{2.4.1} = \left\{ \begin{array}{l} \mathit{animal}(X) \prec \mathit{bird}(X) \\ \mathit{fly}(X) \prec \mathit{bird}(X) \\ \mathit{bird}(X) \prec \mathit{eagle}(X) \\ \mathit{bird}(X) \prec \mathit{penguin}(X) \\ \sim\!\mathit{fly}(X) \prec \mathit{penguin}(X) \end{array} \right\}
$$

Ejemplo 4.3.2 (Continúa el Ejemplo 2.4.4) *Consideremos nuevamente el Ejemplo 2.4.4 presentado en la página 49. Cuando aplicamos la función de traducción $\mathfrak{T}_\Delta$ a la terminología allí presentada obtenemos el siguiente conjunto $\Delta_{2.4.4}$ de reglas rebatibles:*

$$\Delta_{2.4.4} = \left\{ \begin{array}{l} woman(X) \prec marriedWoman(X) \\ \sim divorcee(X) \prec marriedWoman(X) \\ marriedWoman(X) \prec hasHusband(X) \\ hadHusband(X) \prec divorcee(X) \\ \sim hasHusband(X) \prec divorcee(X) \\ marriedWoman(X) \prec hadHusband(X) \end{array} \right\}$$

Ejemplo 4.3.3 (Continúa el Ejemplo 2.4.5) *Consideremos nuevamente el Ejemplo 2.4.5 presentado en la página 50. Cuando aplicamos la función de traducción $\mathfrak{T}_\Delta$ a la terminología allí presentada obtenemos el siguiente conjunto de reglas rebatibles:*

$$\Delta_{2.4.5} = \left\{ \begin{array}{l} centralNervousSystem(X) \prec brain(X) \\ bodyPart(X) \prec brain(X) \\ nervousSystem(X) \prec centralNervousSystem(X) \\ \sim nervousSystem(X) \prec bodyPart(X) \end{array} \right\}$$

4.3.3. Representación de axiomas de subclase e igualdad como reglas estrictas

Ahora, como continuación de nuestro acercamiento al razonamiento con ontologías potencialmente inconsistentes, abordamos la representación de terminologías expresadas en Lógicas para la Descripción como un conjunto de reglas estrictas de la Programación en Lógica Rebatible. Para ello, asumiremos que la terminología a representar en el lenguaje de las reglas estrictas es *consistente*. La presencia de inconsistencias en una terminología involucraría realizar algún tipo de *revisión* (Alchourron et al., 1985); discutiremos tales extensiones al acercamiento propuesto en la Sección 4.7.

A continuación presentamos una definición *preliminar* de la función de DL a reglas estrictas DeLP. Esta correspondencia es aplicada a una terminología y la hace corresponder con un conjunto de reglas estrictas. Luego, veremos las motivaciones que hacen que esto sea insuficiente y presentaremos así una versión definitiva de la misma.

Definición 4.3.8 (Correspondencia $\mathfrak{T}^*_\Pi$ desde sentencias DL hacia reglas estrictas DeLP (versión preliminar)) *Sean A, C, D conceptos, X, Y variables, P, Q relaciones. La correspondencia $\mathfrak{T}^*_\Pi : 2^{\mathcal{L}_{DL}} \rightarrow 2^{\mathcal{L}_{DeLP_\Pi}}$ se define como se presenta en la Figura 4.3. Además, las reglas de la forma "$(H_1 \wedge H_2) \leftarrow B$" son reescritas como dos reglas "$H_1 \leftarrow B$" y "$H_2 \leftarrow B$", las reglas de la forma "$H_1 \leftarrow H_2 \leftarrow B$" son reescritas como "$H_1 \leftarrow B \wedge H_2$", y las reglas de la forma "$H \leftarrow (B_1 \vee B_2)$" son reescritas como dos reglas "$H \leftarrow B_1$" y "$H \leftarrow B_2$".*

De la misma forma que con la función de traducción de sentencias DL a reglas rebatibles, la función $\mathfrak{T}_\Pi$ hace corresponder conjuntos de sentencias DL con conjuntos de reglas estrictas DeLP. Los casos recursivos son dos: (1) el caso en que la entrada es un axioma

$$
\begin{array}{rcl}
\mathfrak{T}^*_\Pi(\{C \sqsubseteq D\}) & =_{def} & \left\{ T_h(D,X) \leftarrow T_b(C,X) \right\}, \\
 & & \text{si } C \text{ es una } \mathcal{L}_b\text{-clase y } D \text{ una } \mathcal{L}_h\text{-clase} \\
\mathfrak{T}^*_\Pi(\{C \equiv D\}) & =_{def} & \mathfrak{T}^*_\Pi(\{C \sqsubseteq D\}) \cup \mathfrak{T}^*_\Pi(\{D \sqsubseteq C\}), \text{ si } C \text{ y } D \text{ son } \mathcal{L}_{hb}\text{-clases} \\
\mathfrak{T}^*_\Pi(\{\top \sqsubseteq \forall P.D\}) & =_{def} & \left\{ T_h(D,Y) \leftarrow P(X,Y) \right\}, \text{ si } D \text{ es una } \mathcal{L}_h\text{-clase} \\
\mathfrak{T}^*_\Pi(\{\top \sqsubseteq \forall P^-.D\}) & =_{def} & \left\{ T_h(D,X) \leftarrow P(X,Y) \right\}, \text{ si } D \text{ es una } \mathcal{L}_h\text{-clase} \\
\mathfrak{T}^*_\Pi(\{a : D\}) & =_{def} & \left\{ T_h(D,a) \right\}, \text{ si } D \text{ es una } \mathcal{L}_h\text{-clase} \\
\mathfrak{T}^*_\Pi(\{\langle a,b\rangle : P\}) & =_{def} & \left\{ P(a,b) \right\} \\
\mathfrak{T}^*_\Pi(\{P \sqsubseteq Q\}) & =_{def} & \left\{ Q(X,Y) \leftarrow P(X,Y) \right\} \\
\mathfrak{T}^*_\Pi(\{P \equiv Q\}) & =_{def} & \left\{ \begin{array}{l} Q(X,Y) \leftarrow P(X,Y) \\ P(X,Y) \leftarrow Q(X,Y) \end{array} \right\} \\
\mathfrak{T}^*_\Pi(\{P \equiv Q^-\}) & =_{def} & \left\{ \begin{array}{l} Q(X,Y) \leftarrow P(Y,X) \\ P(Y,X) \leftarrow Q(X,Y) \end{array} \right\} \\
\mathfrak{T}^*_\Pi(\{P^+ \sqsubseteq P\}) & =_{def} & \left\{ P(X,Z) \leftarrow P(X,Y) \wedge P(Y,Z) \right\} \\
\mathfrak{T}^*_\Pi(\{s_1,\ldots,s_n\}) & =_{def} & \bigcup_{i=1}^{n} \mathfrak{T}^*_\Pi(\{s_i\}), \text{ si } n > 1
\end{array}
$$

donde:

$$
\begin{array}{rcl}
T_h(A,X) & =_{def} & A(X) \\
T_h((C \sqcap D),X) & =_{def} & T_h(C,X) \wedge T_h(D,X) \\
T_h((\forall R.C),X) & =_{def} & T_h(C,Y) \leftarrow R(X,Y) \\
T_b(A,X) & =_{def} & A(X) \\
T_b((C \sqcap D),X) & =_{def} & T_b(C,X) \wedge T_b(D,X) \\
T_b((C \sqcup D),X) & =_{def} & T_b(C,X) \vee T_b(D,X) \\
T_b((\exists R.C),X) & =_{def} & R(X,Y) \wedge T_b(C,Y)
\end{array}
$$

Figura 4.3: Función de traducción de ontologías DL a reglas estrictas DeLP

de igualdad, el cual es interpretado como dos axiomas de inclusión, y (2) el caso en que la entrada es un conjunto arbitrario de al menos dos sentencias.

Debe notarse que la simple traducción de sentencias DL a reglas DeLP estrictas no permite inferir información negativa cuando la misma se halla implícita en la ontología original.

Ejemplo 4.3.4 *Consideremos la ontología* $\Sigma_1 = (T_S, \emptyset, A)$ *donde:*

$$
\begin{array}{rcl}
T_S & = & \left\{ \textit{penguin} \sqsubseteq \neg \textit{flies} \right\}, y \\
A & = & \left\{ \begin{array}{l} \textit{opus} : \textit{penguin}; \\ \textit{tweety} : \textit{flies} \end{array} \right\}.
\end{array}
$$

*Esta ontología es consistente. Si alimentamos a un razonador estándar DL como Racer con esta ontología, seremos capaces de deducir que Opus pertenece a la clase de los no voladores (*i.e., $\neg \textit{flies}(\textit{opus})$*) y que Tweety no es un pingüino (*i.e., $\neg \textit{penguin}(\textit{tweety})$*).*

En cambio, si consideramos al programa DeLP $\mathcal{P} = (\Pi, \Delta)$ *que se obtiene con la función* $\mathfrak{T}^*_\Pi$ *de la Definición 4.3.10 de la página 112:*

$$\begin{array}{rcl} \Pi & = & \left\{ \begin{array}{l} \mathit{penguin}(\mathit{opus}); \\ \mathit{flies}(\mathit{tweety}); \\ \sim\mathit{flies}(X) \leftarrow \mathit{penguin}(X) \end{array} \right\} \\ \Delta & = & \{\}, \end{array}$$

vemos que no es posible deducir que Tweety no es un pingüino.

Además, una vez que tenemos información estricta es necesario considerar la deducción de información negativa. Por ejemplo, la sentencia de inclusión $C \sqsubseteq D$ (todos los C's son D's) es traducida como la regla estricta $D(X) \leftarrow C(X)$. Claramente, a partir de $C(a)$ (*i.e.*, a pertenece al concepto C) podemos deducir $D(a)$ (*i.e.*, a pertenece al concepto D). Sin embargo, en presencia de $\sim D(a)$ (*i.e.*, a no pertenece a D) no es posible deducir $\sim C(a)$ (*i.e.*, a no pertenece a C).

En virtud de ello, dada una sentencia DL $C_1 \sqcap C_2 \sqcap \ldots \sqcap C_{n-1} \sqcap C_n \sqsubseteq D$, en lugar de sólo incluir la única regla estricta $D(X) \leftarrow C_1(X), C_2(X), \ldots, C_{n-1}(X), C_n(X)$ en el programa DeLP derivado de la ontología, proponemos en cambio incluir a todas las *transpuestas* de dicha regla.

Definición 4.3.9 (Transpuestas de una regla estricta) *Sea r una regla estricta tal que $r = H \leftarrow B_1, B_2, B_3, \ldots, B_{n-1}, B_n$. El conjunto de las reglas transpuestas de la regla r, denotado como $\mathfrak{Transp}(r)$, se calcula como:*

$$\mathfrak{Transp}(\cdot) : \mathcal{L}_{DeLP_\Pi} \rightarrow 2^{\mathcal{L}_{DeLP_\Pi}}$$

$$\mathfrak{Transp}(H \leftarrow B_1, B_2, \ldots, B_{n-1}, B_n) = \left\{ \begin{array}{l} H \leftarrow B_1, B_2, \ldots, B_{n-1}, B_n \\ \overline{B_1} \leftarrow \overline{H}, B_2, B_3, \ldots, B_{n-1}, B_n \\ \overline{B_2} \leftarrow \overline{H}, B_1, B_3, \ldots, B_{n-1}, B_n \\ \overline{B_3} \leftarrow \overline{H}, B_1, B_2, \ldots, B_{n-1}, B_n \\ \ldots \\ \overline{B_{n-1}} \leftarrow \overline{H}, B_1, B_2, B_3 \ldots, B_n \\ \overline{B_n} \leftarrow \overline{H}, B_1, B_2, \ldots, B_{n-1} \end{array} \right\}$$

Ejemplo 4.3.5 (Continúa el Ejemplo 4.3.4) *Consideremos nuevamente el Ejemplo 4.3.4 de la página 110. En particular, veamos la regla estricta:*

$$r_1 : \sim\mathit{flies}(X) \leftarrow \mathit{penguin}(X).$$

Al calcular las transpuestas de la regla r_1, i.e. *$\mathfrak{Transp}(r_1)$, obtenemos el conjunto de reglas:*

$$\mathfrak{Transp}(r_1) = \left\{ \begin{array}{l} \sim\mathit{flies}(X) \leftarrow \mathit{penguin}(X) \\ \sim\mathit{penguin}(X) \leftarrow \mathit{flies}(X) \end{array} \right\}.$$

*Así, en presencia de la información asercional que establece que Opus es un pingüino (*i.e., *penguin*(*opus*)) *es posible derivar que Opus no es capaz de volar (*i.e., $\sim$*flies*(*opus*)). *Además, ahora en presencia de la información derivada que expresa que Tweety es capaz de volar (*i.e., *flies*(*tweety*)), *podremos deducir que Tweety no es un pingüino (*i.e., $\sim$ *penguin*(*tweety*)).

Definición 4.3.10 (Correspondencia $\mathfrak{T}_\Pi$ desde sentencias DL hacia reglas estrictas DeLP) *Sea $\mathfrak{T}^*_\Pi$ la función de traducción de terminologías DL a reglas estrictas presentadas en la Definición 4.3.8. La correspondencia que toma una terminología DL T consistente y la hace corresponder con un conjunto de reglas estrictas, $\mathfrak{T}_\Pi : 2^{\mathcal{L}_{DL}} \rightarrow 2^{\mathcal{L}_{DeLP_\Pi}}$, se define como:*

$$\mathfrak{T}_\Pi(T) = \mathfrak{Transp}(\mathfrak{T}^*_\Pi(T)).$$

Nótese también que, siguiendo las tendencias en razonamiento nomonótono, no consideramos la transposición de reglas rebatibles (Brewka et al., 1997; Caminada, 2008). En este respecto pero en el contexto la Lógica Default, Brewka et al. (1997, pág. 45) afirman que:

> El poder expresivo de la Lógica Default se debe principalmente a la representación de los *defaults* como reglas (no estándar) de inferencia. Esta representación evita problemas con la transposición de los *defaults*. En la lógica clásica, una implicación lógica $A \supset B$ es equivalente a su contraposición $\neg B \supset \neg A$. Para los *defaults*, la contraposición es a veces indeseada. Por ejemplo, uno podría suponer que los científicos de la computación típicamente no saben mucho acerca del razonamiento nomonótono. A partir de esta creencia, ciertamente no se deduce que aquellos que saben mucho acerca del razonamiento nomonótono no son típicamente científicos de la computación.

Ejemplo 4.3.6 (Adaptado de (Baader et al., 2003)) *Consideremos la siguiente terminología T donde hemos etiquetado a los axiomas de la misma como $r_i, i = 1, \ldots, 4$:*

$$T = \left\{ \begin{array}{lcl} r_1 & : & \mathit{Woman} \equiv \mathit{Female} \\ r_2 & : & \mathit{Man} \equiv \mathit{Person} \sqcap \neg \mathit{Woman} \\ r_3 & : & \mathit{Woman} \sqcap \exists \mathit{hasChild}.\mathit{Person} \sqsubseteq \mathit{Mother} \\ r_4 & : & \mathit{Father} \sqcup \mathit{Mother} \sqsubseteq \mathit{Parent} \end{array} \right\}$$

Calculemos primero el resultado de aplicar $\mathfrak{T}^*_\Pi$ a T, como cada axioma de T genera uno o más de una regla estricta en $\mathfrak{T}^*_\Pi(T)$, hemos etiquetado a las reglas estrictas obtenidas a partir del axioma r_i como $r_i^1, \ldots, r_i^{n_i}$ con $i = 1, \ldots, 4$:

$$\mathfrak{T}^*_{\Pi}(T) = \left\{ \begin{array}{lcl} r_1^1 & : & woman(X) \leftarrow female(X) \\ r_1^2 & : & female(X) \leftarrow woman(X) \\ r_2^1 & : & person(X) \leftarrow man(X) \\ r_2^2 & : & \sim woman(X) \leftarrow man(X) \\ r_2^3 & : & man(X) \leftarrow person(X), \sim woman(X) \\ r_3^1 & : & mother(X) \leftarrow woman(X), hasChild(X,Y), person(Y) \\ r_4^1 & : & parent(X) \leftarrow father(X) \\ r_4^2 & : & parent(X) \leftarrow mother(X) \end{array} \right\}$$

Ahora, al calcular las transpuestas de cada una de las reglas de $\mathfrak{T}^*_{\Pi}(T)$*, hemos etiquetado a las reglas* r_i^j *del conjunto* $\mathfrak{T}^*_{\Pi}(T)$ *como* $r_i^j.(a), r_i^j.(b), \ldots$*, para obtener:*

$$\mathfrak{T}_{\Pi}(T) = \left\{ \begin{array}{lcl} r_1^1.(a) & : & woman(X) \leftarrow female(X) \\ r_1^1.(b) & : & \sim female(X) \leftarrow \sim woman(X) \\ & & \\ r_1^2.(a) & : & female(X) \leftarrow woman(X) \\ r_1^2.(b) & : & \sim woman(X) \leftarrow \sim female(X) \\ & & \\ r_2^1.(a) & : & person(X) \leftarrow man(X) \\ r_2^1.(b) & : & \sim man(X) \leftarrow \sim person(X) \\ & & \\ r_2^2.(a) & : & \sim woman(X) \leftarrow man(X) \\ r_2^2.(b) & : & \sim man(X) \leftarrow woman(X) \\ & & \\ r_2^3.(a) & : & man(X) \leftarrow person(X), \sim woman(X) \\ r_2^3.(b) & : & \sim person(X) \leftarrow \sim man(X), \sim woman(X) \\ r_2^3.(c) & : & woman(X) \leftarrow person(X), \sim man(X) \\ & & \\ r_3^1.(a) & : & mother(X) \leftarrow woman(X), hasChild(X,Y), person(Y) \\ r_3^1.(b) & : & \sim woman(X) \leftarrow \sim mother(X), hasChild(X,Y), person(Y) \\ r_3^1.(c) & : & \sim hasChild(X,Y) \leftarrow woman(X), \sim mother(X), person(Y) \\ r_3^1.(d) & : & \sim person(Y) \leftarrow woman(X), hasChild(X,Y), \sim mother(X) \\ & & \\ r_4^1.(a) & : & parent(X) \leftarrow father(X) \\ r_4^1.(b) & : & \sim father(X) \leftarrow \sim parent(X) \\ & & \\ r_4^2.(a) & : & parent(X) \leftarrow mother(X) \\ r_4^2.(b) & : & \sim mother(X) \leftarrow \sim parent(X) \end{array} \right\}.$$

Ejemplo 4.3.7 (Continúa el Ejemplo 4.3.6) *Consideremos la Abox A definida respecto a la terminología presentada en el Ejemplo 4.3.6:*

$$A = \left\{ \begin{array}{l} marta : Mother \\ carlos : Father \\ marta : Female \\ carlos : Man \\ \langle carlos, sergio \rangle : hasChild \\ \langle marta, sergio \rangle : hasChild \\ sergio : Person \end{array} \right\}.$$

Cuando aplicamos la función $\mathfrak{T}_\Pi$ a A, obtenemos:

$$\mathfrak{T}_\Pi(A) = \left\{ \begin{array}{l} mother(marta) \\ father(carlos) \\ female(marta) \\ man(carlos) \\ hasChild(carlos, sergio) \\ hasChild(marta, sergio) \\ person(sergio) \end{array} \right\}.$$

Tratamiento de $\top$ y $\bot$ en DeLP: El símbolo $\top$ se trata como un identificador de concepto llamado *true* mientras que $\bot$ se trata como un identificador de concepto llamado *false*. Se realiza la traducción como se explicó en las secciones previas. Todas las apariciones de *true* en el cuerpo de las reglas se eliminan (ya que siempre es satisfecho). Todas las reglas que contengan apariciones de *false* en el cuerpo se eliminan (ya que nunca van a ser satisfechas). Las apariciones de true y false en la cabeza de las reglas no tienen sentido y estas reglas se eliminan.

4.4. Ontologías basadas en argumentación rebatible

Como hemos mencionado anteriormente, una ontología es una representación de una conceptualización de una parte relevante de un dominio de aplicación. Las Lógicas para la Descripción (DL) proveen un formalismo elegante para realizar tal representación. Si bien existen razonadores eficientes para realizar inferencias a partir de ontologías definidas en DL, éstos no funcionan en forma satisfactoria en presencia de ontologías inconsistentes.

En esta sección presentamos un formalismo donde las posibles inconsistencias en una ontología serán expresadas como un conjunto de axiomas de inclusión de clases que serán considerados *rebatibles*. De esta manera, en presencia de información incompleta y posiblemente inconsistente, realizaremos un análisis dialéctico que considerará todas las razones a favor y en contra para determinar el estado epistémico de una afirmación sobre la pertenencia de un individuo a una determinada clase.

4.4.1. Representación de conocimiento: δ-ontologías

Una ontología define un conjunto de clases y un conjunto de individuos perteneciendo a dichas clases. Redefinimos la noción de ontología DL para adecuarla a nuestro acercamiento.

Definición 4.4.1 (δ-Ontología) *Sean C una clase-$\mathcal{L}_b$, D una clase $\mathcal{L}_h$, A, B $\mathcal{L}$-clases, P, Q propiedades, a, b individuos. Sea T un conjunto de sentencias de inclusión y equivalencia de $\mathcal{L}_{DL}$ de la forma "$C \sqsubseteq D$", "$A \equiv B$", "$\top \sqsubseteq \forall P.D$", "$\top \sqsubseteq \forall P^{-}.D$", "$P \sqsubseteq Q$", "$P \equiv Q$", "$P \equiv Q^{-}$", o "$P^{+} \sqsubseteq P$" tal que T se puede particionar en dos conjuntos disjuntos T_S y T_D (*i.e., *$T_S \cup T_D = T$ y $T_S \cap T_D = \emptyset$). Sea A un conjunto de aserciones disjunto con T de la forma $a : D$ o $\langle a, b \rangle : P$. Una δ-ontología Σ es una tupla (T_S, T_D, A). Llamaremos al conjunto T_S la terminología estricta (o Sbox); el conjunto T_D será llamado la terminología rebatible (o Dbox), y el conjunto A será llamado la caja asercional (o Abox).*

Veremos que más adelante daremos una semántica para estas ontologías basada en que las mismas pueden ser expresadas en el lenguaje de la Programación en Lógica Rebatible (DeLP). La restricción de la Tbox de poder ser particionada permitirá que el conjunto Sbox de sentencias estrictas de inclusión pueda ser interpretado como un conjunto de reglas estrictas en DeLP mientras que el conjunto Dbox de reglas tentativas de inclusión será interpretado como un conjunto de reglas rebatibles en DeLP.

Ejemplo 4.4.1 *Sea la siguiente δ-ontología $\Sigma_{4.4.1} = (T_S, T_D, A)$, donde:* [6]

$$T_S = \left\{ \begin{array}{l} r_1 : \mathit{chicken} \sqsubseteq \mathit{bird} \\ r_2 : \mathit{penguin} \sqsubseteq \mathit{bird} \\ r_3 : \mathit{penguin} \sqsubseteq \neg \mathit{flies} \end{array} \right\}$$

$$T_D = \left\{ \begin{array}{l} r_4 : \mathit{bird} \sqsubseteq \mathit{flies} \\ r_5 : \mathit{chicken} \sqsubseteq \neg \mathit{flies} \\ r_6 : \mathit{chicken} \sqcap \mathit{scared} \sqsubseteq \mathit{flies} \\ r_7 : \mathit{flies} \sqsubseteq \mathit{nest\_in\_trees} \end{array} \right\}$$

$$A = \left\{ \begin{array}{l} r_8 : \mathit{tina} : \mathit{chicken} \\ r_9 : \mathit{penguin} : \mathit{tweety} \\ r_{10} : \mathit{tina} : \mathit{scared} \end{array} \right\}$$

La terminología estricta se presenta en el conjunto T_S. El axioma r_1 establece que las gallinas son aves, mientras que r_2 dice que los pingüinos son aves, r_3 expresa que los pingüinos no vuelan. La terminología rebatible se presenta en el conjunto T_D. El axioma

[6] Este ejemplo aparece en (García and Simari, 2004) en un contexto diferente.

de inclusión rebatible r_4 indica que las aves usualmente vuelan, r_5 dice que las gallinas usualmente no vuelan, r_6 dice que las gallinas que además están asustadas usualmente son capaces de volar y r_7 indica que los animales voladores usualmente anidan en árboles. La caja asercional se presenta en el conjunto A. r_8 expresa que Tina es una gallina, r_9 dice que Tweety es un pingüino y r_{10} establece que Tina está asustada.

Ejemplo 4.4.2 (Diamante de Nixon) *Consideremos el problema del diamante de Nixon (Prakken and Vreeswijk, 2002). Sea $\Sigma_{4.4.2} = (\emptyset, T_D, A)$ una δ-ontología donde:* [7]

$$T_D = \left\{ \begin{array}{l} lives\_in\_chicago \sqsubseteq has\_a\_gun \\ lives\_in\_chicago \sqcap pacifist \sqsubseteq \neg has\_a\_gun \\ quaker \sqsubseteq pacifist \\ republican \sqsubseteq \neg pacifist \end{array} \right\}$$

$$A = \left\{ \begin{array}{l} nixon : lives\_in\_chicago \\ nixon : quaker \\ nixon : republican \end{array} \right\}$$

Si alguien vive en Chicago, entonces usualmente tiene un arma. Si alguien vive en Chicago y es un pacifista, entonces usualmente no tiene un arma. Los quákeros son usualmente pacifistas. Los republicanos no son usualmente pacifistas. Nixon vive en Chicago, es un quákero y es republicano.

Ejemplo 4.4.3 (Aplicación comercial) *Consideremos la siguiente δ-ontología $\Sigma_{4.4.3} = (\emptyset, T_D, A)$ perteneciente al dominio de los negocios:* [8]

$$T_D = \left\{ \begin{array}{l} good\_price \sqsubseteq buy\_stock \\ good\_price \sqcap risky\_company \sqsubseteq \neg buy\_stock \\ \exists in\_fusion.\top \sqcup closing \sqsubseteq risky\_company \\ \exists in\_fusion.strong \sqsubseteq \neg risky\_company \end{array} \right\}$$

$$A = \left\{ \begin{array}{l} acme : good\_price \\ \langle acme, steel \rangle : in\_fusion \\ steel : strong \end{array} \right\}.$$

El significado de la ontología es el siguiente. Si las acciones de una empresa tienen buen precio entonces es usualmente razonable comprar acciones de tal empresa a menos que la compañía sea riesgosa. Las compañías riesgosas son usualmente aquellas que se hallan en proceso de fusión o cerrando. Sin embargo, si una compañía se halla en fusión con una compañía fuerte, entonces usualmente no es una compañía riesgosa. Se conoce que Acme tiene acciones a buen precio, que Acme está en fusión con Steel, la cual es una compañía fuerte.

[7] Este ejemplo aparece en (García and Simari, 2004) en un contexto diferente.
[8] Este ejemplo aparece en (García and Simari, 2004) en un contexto diferente.

Ejemplo 4.4.4 (Elefantes reales africanos) *Sea la siguiente δ-ontología* $\Sigma_{4.4.4} = (T_S, T_D, A)$ *donde:* [9]

$$T_D = \left\{ \begin{array}{l} elephant \sqsubseteq gray \\ royal\_elephant \sqsubseteq \neg gray \end{array} \right\}$$

$$T_S = \left\{ \; royal\_elephant \sqcup african\_elephant \sqsubseteq elephant \; \right\}$$

$$A = \left\{ \begin{array}{l} clyde : royal\_elephant \\ clyde : african\_elephant \end{array} \right\}.$$

La interpretación de la ontología es como sigue: Los elefantes tienden a ser grises, los elefantes reales tienden a no ser grises, los elefantes reales son elefantes, los elefantes africanos son elefantes, Clyde es un elefante real, Clyde es un elefante africano.

Ejemplo 4.4.5 (Estudiantes universitarios adultos) *Sea la siguiente ontología* $\Sigma_{4.4.5} = (\emptyset, T_D, A)$ *donde:* [10]

$$T_D = \left\{ \begin{array}{l} adult \sqsubseteq worker \\ university\_student \sqsubseteq \neg worker \\ university\_student \sqsubseteq adult \end{array} \right\}$$

$$A = \left\{ \begin{array}{l} ken : adult \\ ken : university\_student \end{array} \right\}.$$

El significado de la ontología es como sigue: Los adultos son usualmente trabajadores. Los estudiantes universitarios usualmente no trabajan. Los estudiantes universitarios son usualmente adultos. Ken es un adulto y un estudiante universitario.

Ejemplo 4.4.6 (Boxeadores violentos) *Sea la siguiente δ-ontología* $\Sigma_{4.4.6} = (\emptyset, T_D, A)$ *tal que:* [11]

$$T_D = \left\{ \begin{array}{l} boxer \sqsubseteq violent \\ boxer \sqcap animal\_lover \sqsubseteq \neg violent \\ \exists loves.airedale \sqsubseteq animal\_lover \\ \exists kicks.poodle \sqsubseteq \neg animal\_lover \end{array} \right\}, \; y$$

$$A = \left\{ \begin{array}{l} john : boxer \\ \langle john, lola \rangle : loves \\ lola : airedale \\ \langle john, tina \rangle : kicks \\ tina : poodle \end{array} \right\}$$

[9] Este ejemplo aparece en (Simari and Loui, 1992) en un contexto diferente.
[10] Este ejemplo aparece en (Simari and Loui, 1992) en un contexto diferente.
[11] Este ejemplo fue presentado en (Gómez et al., 2008*c*).

La terminología rebatible T_D expresa que: los boxeadores son violentos; un boxeador que es amante de los animales no es violento; si alguien ama a un perro de raza Airedale Terrier es un amante de los animales, y, si alguien es capaz de patear a un Poodle, entonces no es un amante de los animales. La Abox A expresa que: John es boxeador y ama a Lola; Lola es un Airedale Terrier; John patea a Tina y Tina es un Poodle.

La ontología presentada en el ejemplo anterior es claramente inconsistente ya que desde el punto de los razonadores DL estándar la clase de los boxeadores debería ser vacía. Esto se debe a que es posible demostrar que un boxeador es violento y no violento al mismo tiempo (*i.e.*, $boxer \sqsubseteq violent \sqcap \neg violent = \bot$), y por otro lado se conoce un boxeador llamado John (*i.e.*, $boxer(john)$).

Ejemplo 4.4.7 (Animales modificados genéticamente) *Sea la siguiente δ-ontología $\Sigma_{4.4.7} = (\emptyset, T_D, A)$ donde:* [12]

$$T_D = \left\{ \begin{array}{l} s_1 : bird \sqsubseteq animal; \\ s_2 : bird \sqsubseteq fly; \\ s_3 : eagle \sqsubseteq bird; \\ s_4 : penguin \sqsubseteq bird; \\ s_5 : penguin \sqsubseteq \neg fly; \\ s_6 : penguin \sqcap \exists isOperatedBy.geneticSurgeon \\ \qquad\qquad\qquad\qquad \sqsubseteq geneticallyAlteredPenguin; \\ s_7 : geneticallyAlteredPenguin \sqsubseteq fly; \\ s_8 : eagle \sqcap hasBrokenWing \sqsubseteq \neg fly; \\ s_9 : isOperatedBy \equiv operates^{-}; \\ s_{10} : geneticSurgeon \equiv geneticist \sqcap surgeon \end{array} \right\}$$

$$A = \left\{ \begin{array}{l} s_{11} : opus : penguin \\ s_{12} : avenger : eagle \\ s_{13} : avenger : hasBrokenWing \\ s_{14} : frankenstein : geneticist \\ s_{15} : frankenstein : surgeon \\ s_{16} : \langle frankenstein, opus \rangle : operates \end{array} \right\}$$

La sentencia s_1 en la Tbox T_D dice que cada ave es un animal, la sentencia s_2 dice que cada ave vuela. Las sentencias s_3 y s_4 dicen que las aves y los pingüinos son aves. Las sentencias s_5, s_6 y s_7 dicen que los pingüinos no vuelan a menos que hayan sido genéticamente alterados, donde un pingüino genéticamente alterado es un pingüino que ha sido operado por un cirujano genético. La sentencia s_8 establece que las águilas con las alas rotas no son capaces de volar. La sentecia s_9 define la relación isOperatedBy como la inversa de la relación operates. Finalmente, la sentencia s_{10} define a un cirujano genético

[12] Este ejemplo fue presentado en (Gómez et al., 2007*b*).

como un genetista que es a la vez un cirujano. La sentencia s_{11} en la Abox A expresa que Opus es un pingüino. Las sentencias s_{12} y s_{13} establecen que Avenger es un águila que tiene un ala rota. Las sentencias s_{14} y s_{15} dicen que Frankenstein es a la vez un genetista y un cirujano, respectivamente. Finalmente, la sentencia s_{16} establece que Frankenstein ha operado a Opus.

Ejemplo 4.4.8 *Considere la δ-ontología $\Sigma_{4.4.8} = (T_S, T_D, A)$ basada en el programa DeLP presentado en el Ejemplo 3.3.8 (página 73):*

$$T_S = \left\{ checks\_web\_mail \sqsubseteq uses\_browser \right\}$$

$$T_D = \left\{ \begin{array}{l} studies \sqsubseteq pass \\ seats\_in\_computer \sqsubseteq studies \\ seats\_in\_computer \sqcap web\_surfing \sqsubseteq \neg studies \\ \neg studies \sqsubseteq \neg pass \\ uses\_browser \sqsubseteq web\_surfing \\ uses\_browser \sqcap reads\_javadoc \sqsubseteq \neg web\_surfing \end{array} \right\}$$

$$A = \left\{ \begin{array}{l} john : seats\_in\_computer \\ paul : seats\_in\_computer \\ paul : uses\_browser \\ mary : seats\_in\_computer \\ mary : checks\_web\_mail \\ mary : reads\_javadoc \end{array} \right\}$$

Ejemplo 4.4.9 *Considere la ontología presentada anteriormente en el Ejemplo 2.4.3. Modelamos tal ontología DL como la δ-ontología $\Sigma_{4.4.9} = (T_S, T_D, A)$ donde:*

$$\begin{array}{rcl} T_S & = & \left\{ B \equiv \neg D \right\} \\ T_D & = & \left\{ \begin{array}{l} A \sqsubseteq B \sqcap C \\ A \sqsubseteq D \end{array} \right\} \\ A & = & \left\{ x : A \right\} \end{array}$$

4.4.2. Semántica de δ-ontologías como programas DeLP

En la sección anterior hemos visto diversos ejemplos de δ-ontologías. En dichos ejemplos, las terminologías con posibles inconsistencias han sido definidas como *rebatibles* para diferenciarlas de aquellas terminologías de las cuales el ingeniero de conocimiento garantiza la ausencia de inconsistencias, que por su parte son llamadas *estrictas*. Por otro, lado, la información sobre los individuos se llama *caja asercional* y se supone consistente internamente y, a su vez, consistente con la terminología estricta.

Como hemos mencionado anteriormente, el acercamiento tradicional a la asignación de semántica a las ontologías DL se realiza con semántica basada en teoría de modelos. El problema de tal enfoque reside en que cuando las ontologías involucradas son inconsistentes, como las lógicas DL son un subconjunto de la Lógica de Primer Orden, el mecanismo de inferencia tiene un efecto explosivo ya que cualquier sentencia se convierte en demostrable a partir de la ontología en cuestión. En esta tesis se propone un acercamiento argumentativo al tratamiento de las inconsistencias en ontologías posiblemente inconsistentes. Ahora, veremos cómo asignar semántica a las δ-ontologías por medio de un programa lógico rebatible.

Definición 4.4.2 (Semántica de una δ-ontología) *Sea $\Sigma = (T_S, T_D, A)$ una δ-ontología. La semántica de la δ-ontología Σ es un programa DeLP $\mathcal{P}$ tal que:*

$$\mathcal{P} = \left(\ \mathfrak{T}_{\Pi}(T_S) \cup \mathfrak{T}_{\Pi}(A), \mathfrak{T}_{\Delta}(T_D) \ \right).$$

Cuando corresponda, sintéticamente, notaremos al programa $\mathcal{P}$ como $\mathfrak{T}(\Sigma)$.

A continuación mostraremos diversos ejemplos de cómo las δ-ontologías son interpretadas como programas DeLP.

Ejemplo 4.4.10 (Continúa el Ejemplo 4.4.1) *Considere nuevamente la δ-ontología $\Sigma_{4.4.1} = (T_S, T_D, A)$. La semántica de esta ontología está dada por el programa DeLP $\mathcal{P}_{4.4.1} = (\Pi, \Delta)$, donde:*

$$\Pi = \left\{ \begin{array}{l} bird(X) \leftarrow chicken(X) \\ \sim chicken(X) \leftarrow \sim bird(X) \\ bird(X) \leftarrow penguin(X) \\ \sim penguin(X) \leftarrow \sim bird(X) \\ \sim flies(X) \leftarrow penguin(X) \\ \sim penguin(X) \leftarrow flies(X) \\ chicken(tina) \\ penguin(tweety) \\ scared(tina) \end{array} \right\}$$

$$\Delta = \left\{ \begin{array}{l} \sim flies(X) \prec chicken(X) \\ flies(X) \prec chicken(X), scared(X) \\ nest\_in\_trees(X) \prec flies(X) \end{array} \right\}$$

Ejemplo 4.4.11 (Continúa el Ejemplo 4.4.2) *Consideremos nuevamente la δ-ontologia $\Sigma_{4.4.2} = (\emptyset, T_D, A)$, la semántica de esta ontología es el programa DeLP $\mathcal{P}_{4.4.2} = (\Pi, \Delta)$ donde:*

$$\Pi = \left\{ \begin{array}{l} lives\_in\_chicago(nixon) \\ quaker(nixon) \\ republican(nixon) \end{array} \right\}$$

$$\Delta = \left\{ \begin{array}{l} has\_a\_gun(X) \prec lives\_in\_chicago(X) \\ \sim has\_a\_gun(X) \prec lives\_in\_chicago(X), pacifist(X) \\ pacifist(X) \prec quaker(X) \\ \sim pacifist(X) \prec republican(X) \end{array} \right\}.$$

Ejemplo 4.4.12 (Continúa el Ejemplo 4.4.3) *Consideremos nuevamente la δ-ontología $\Sigma_{4.4.3} = (\emptyset, T_D, A)$, ésta es expresada como el programa DeLP $\mathcal{P}_{4.4.3} = (\Pi, \Delta)$:* [13]

$$\Pi = \left\{ \begin{array}{l} good\_price(acme) \\ in\_fusion(acme, steel) \\ strong(steel) \end{array} \right\}$$

$$\Delta = \left\{ \begin{array}{l} buy\_stock(X) \prec good\_price(X) \\ \sim buy\_stock(X) \prec good\_price(X), risky\_company(X) \\ risky\_company(X) \prec in\_fusion(X, Y) \\ risky\_company(X) \prec closing(X) \\ \sim risky\_company(X) \prec in\_fusion(X, Y), strong(Y) \end{array} \right\}.$$

Ejemplo 4.4.13 (Continúa el Ejemplo 4.4.4) *Considérese nuevamente la δ-ontología $\Sigma_{4.4.4} = (T_S, T_D, A)$. Esta ontología es interpretada como el programa DeLP $\mathcal{P}_{4.4.4} = (\Pi, \Delta)$ donde:*

$$\Pi = \left\{ \begin{array}{l} elephant(X) \leftarrow royal\_elephant(X) \\ \sim royal\_elephant(X) \leftarrow \sim elephant(X) \\ elephant(X) \leftarrow african\_elephant(X) \\ \sim african\_elephant(X) \leftarrow \sim elephant(X) \\ royal\_elephant(clyde) \\ african\_elephant(clyde) \end{array} \right\}$$

$$\Delta = \left\{ \begin{array}{l} gray(X) \prec elephant(X) \\ \sim gray(X) \prec royal\_elephant(X) \end{array} \right\}.$$

Ejemplo 4.4.14 (Continúa el Ejemplo 4.4.5) *Veamos nuevamente la ontología $\Sigma_{4.4.5} = (\emptyset, T_D, A)$ que va a ser interpretada como el programa $\mathcal{P}_{4.4.5} = (\Pi, \Delta)$:*

$$\Pi = \left\{ \begin{array}{l} adult(ken) \\ university\_student(ken) \end{array} \right\}$$

$$\Delta = \left\{ \begin{array}{l} worker(X) \prec adult(X) \\ \sim worker(X) \prec university\_student(X) \\ adult(X) \prec university\_student(X) \end{array} \right\}.$$

[13] Este programa aparece en (García and Simari, 2004).

Ejemplo 4.4.15 (Continúa el Ejemplo 4.4.6) *Considérese nuevamente la δ-ontología* $\Sigma_{4.4.6} = (\emptyset, T_D, A)$*; esta ontología es interpretada como el programa DeLP* $\mathcal{P}_{4.4.6} = (\Pi, \Delta)$ *tal que:*

$$\Pi = \left\{ \begin{array}{l} boxer(john) \\ loves(john, lola) \\ airedale(lola) \\ kicks(john, tina) \\ poodle(tina) \end{array} \right\}, y$$

$$\Delta = \left\{ \begin{array}{l} violent(X) \prec boxer(X) \\ \sim violent(X) \prec boxer(X), animal\_lover(X) \\ animal\_lover(X) \prec loves(X, Y), airedale(Y) \\ \sim animal\_lover(X) \prec kicks(X, Y), poodle(Y) \end{array} \right\}.$$

Ejemplo 4.4.16 (Continúa el Ejemplo 4.4.7) *Considérese nuevamente la δ-ontología* $\Sigma_{4.4.7} = (\emptyset, T_D, A)$*; ésta será interpretada como el programa DeLP* $\mathcal{P}_{4.4.7} = (\Pi, \Delta)$ *tal que:*

$$\Pi = \left\{ \begin{array}{ll} r_{11}: & penguin(opus) \\ r_{12}: & eagle(avenger) \\ r_{13}: & hasBrokenWing(avenger) \\ r_{14}: & geneticist(frankenstein) \\ r_{15}: & surgeon(frankenstein) \\ r_{16}: & operates(frankenstein, opus) \end{array} \right\}, y$$

$$\Delta = \left\{ \begin{array}{ll} r_1: & animal(X) \prec bird(X) \\ r_2: & fly(X) \prec bird(X) \\ r_3: & bird(X) \prec eagle(X) \\ r_4: & bird(X) \prec penguin(X) \\ r_5: & \sim fly(X) \prec penguin(X) \\ r_6: & geneticallyAlteredPenguin(X) \prec \\ & \qquad penguin(X), isOperatedBy(X, Y), geneticSurgeon(Y) \\ r_7: & fly(X) \prec geneticallyAlteredPenguin(X) \\ r_8: & \sim fly(X) \prec eagle(X), hasBrokenWing(X) \\ r_9^1: & operates(X, Y) \prec isOperatedBy(Y, X) \\ r_9^2: & isOperatedBy(X, Y) \prec operates(Y, X) \\ r_{10}^1: & geneticSurgeon(X) \prec \\ & \qquad geneticist(X), surgeon(X) \\ r_{10}^2: & geneticist(X) \prec geneticSurgeon(X) \\ r_{10}^3: & surgeon(X) \prec geneticSurgeon(X) \end{array} \right\}.$$

Cuando por cada sentencia s_i *(con* $i = 1, \ldots, 10$*) perteneciente a la terminología* T_D *definida en la ontología* $\Sigma_{4.4.7}$*, es posible obtener* n_i *reglas rebatibles (con* $n_i > 0$*), hemos etiquetado con* r_i^j *(con* $j = 1, \ldots, n_i$*) a dichas reglas rebatibles; en otro caso, si a partir*

de una sentencia s_i se obtiene sólo una regla, entonces ésta última es simplemente notada como r_i.

Ejemplo 4.4.17 (Continúa el Ejemplo 4.4.8) *El siguiente programa DeLP $\mathcal{P}_{4.4.8} = (\Pi, \Delta)$ es la interpretación de la δ-ontología $\Sigma_{4.4.8}$ con:*

$$\Pi = \left\{ \begin{array}{l} seats\_in\_computer(john) \\ seats\_in\_computer(paul) \\ uses\_browser(paul) \\ seats\_in\_computer(mary) \\ checks\_web\_mail(mary) \\ reads\_javadoc(mary) \\ uses\_browser(X) \leftarrow checks\_web\_mail(X) \\ \sim checks\_web\_mail(X) \leftarrow \sim uses\_browser(X) \end{array} \right\}, y$$

$$\Delta = \left\{ \begin{array}{l} pass(X) \prec studies(X) \\ studies(X) \prec seats\_in\_computer(X) \\ \sim studies(X) \prec seats\_in\_computer(X), web\_surfing(X) \\ \sim pass(X) \prec \sim studies(X) \\ web\_surfing(X) \prec uses\_browser(X) \\ \sim web\_surfing(X) \prec uses\_browser(X), reads\_javadoc(X) \end{array} \right\}.$$

Ejemplo 4.4.18 (Continúa el Ejemplo 4.4.9) *La interpretación de la δ-ontología $\Sigma_{4.4.9}$ es el siguiente programa DeLP $\mathcal{P}_{4.4.9} = (\Pi, \Delta)$ donde:*

$$\Pi = \left\{ \begin{array}{l} \sim d(X) \leftarrow b(X) \\ \sim b(X) \leftarrow d(X) \\ a(x) \end{array} \right\}, y$$

$$\Delta = \left\{ \begin{array}{l} b(X) \prec a(X) \\ c(X) \prec a(X) \\ d(X) \prec a(X) \end{array} \right\}.$$

4.5. Tareas de inferencia en δ-ontologías

Ya hemos mostrado cómo las ontologías inconsistentes no pueden ser procesadas por razonadores estándar. Luego, para resolver el problema del razonamiento con ontologías DL potencialmente inconsistentes haremos una inmersión a la Programación en Lógica Rebatible (DeLP). En este marco de trabajo, hemos visto cómo representar en el lenguaje de la DeLP a un cierto subconjunto de las ontologías definibles en las Lógicas para la Descripción (DL). Ahora, en esta sección, veremos cómo resolver las tareas típicas de inferencia en Aboxes propuestas por las DL en un marco argumentativo.

En el marco DL, luego de que el ingeniero de conocimiento ha diseñado la terminología y ha usado los servicios del sistema DL para chequear que todos los conceptos son

satisfacibles y que valen las relaciones de subsunción esperadas, la Abox se puede llenar con aserciones acerca de individuos. Nótese que estas restricciones no son necesarias en el marco de la DeLP pues, en el mismo, las inconsistencias en la representación del conocimiento, son una característica deseable para poder desarrollar el análisis dialéctico en el que se basa su semántica. Sin embargo, para evitar la formación de argumentos autoderrotantes (Prakken and Vreeswijk, 2002), DeLP exige que la información estricta de un programa rebatible posea cierta coherencia interna. Por lo tanto, hemos de exigir que la caja asercional tenga también una cierta coherencia interna y que, por su parte, la misma sea consistente con la terminología estricta.

Definición 4.5.1 (Coherencia interna en cajas asercionales) *Sea $\Sigma = (T_S, T_D, A)$ una δ-ontología. Sean C un nombre de clase, R un nombre de propiedad, a, b nombres de individuos definidos en Σ. Diremos que A es internamente coherente si y sólo si A no posee al mismo tiempo dos aserciones de la forma $a : C$ y $a : \neg C$. Si A no es internamente coherente se dirá internamente incoherente.*

Definición 4.5.2 (Consistencia de la caja asercional respecto de la terminología estricta) *Sea $\Sigma = (T_S, T_D, A)$ una δ-ontología. Sean C un nombre de clase, R un nombre de propiedad, a, b nombres de individuos definidos en Σ. Diremos que A es consistente respecto de T_S si y sólo si no es posible derivar estrictamente a partir de $\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A)$ dos literales complementarios de la forma $C(a)$ y $\sim C(a)$ al mismo tiempo. Si no ocurre que A es consistente respecto de T_S diremos que A es inconsistente respecto de T_S.*

Ejemplo 4.5.1 *Sea $\Sigma_{4.5.1} = (T_S, T_D, A)$ una δ-ontología tal que:*

$$\begin{aligned} T_S &= \left\{ \begin{array}{l} C \sqsubseteq D \\ D \sqsubseteq \neg F \end{array} \right\}, \text{ y} \\ A &= \left\{ \begin{array}{l} a : C \\ a : F \end{array} \right\}. \end{aligned}$$

En la δ-ontología $\Sigma_{4.5.1}$ expresada como el programa DeLP $\mathcal{P}_{4.5.1} = (\Pi, \Delta)$, tal que

$$\Pi = \mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A) = \left\{ \begin{array}{l} c(a) \\ f(a) \\ d(X) \leftarrow c(X) \\ \sim c(X) \leftarrow \sim d(X) \\ \sim f(X) \leftarrow d(X) \\ \sim d(X) \leftarrow f(X) \end{array} \right\} \text{ y}$$

$$\Delta = \mathfrak{T}_\Delta(T_D),$$

es posible derivar estrictamente a los literales $f(a)$ y $\sim f(a)$. Luego, la δ-ontología $\Sigma_{4.5.1}$ no es consistente.

En la Sección 2.3.2 de la página 36, vimos cuáles son las tareas de razonamiento para ontologías pobladas[14] en el marco de las Lógicas de la Descripción cuando las ontologías en cuestión son consistentes. En el resto de esta sección, veremos cómo definir las tareas de razonamiento con individuos en el marco de la Programación en Lógica Rebatible. Primeramente, en la Sección 4.5.1, veremos cómo implementar el chequeo de instancia. Además, estudiaremos la definición del problema de la recuperación en la Sección 4.5.2.

4.5.1. Chequeo de instancia (*Instance checking*)

En la Sección 2.3.2 (página 36) presentamos la tarea de razonamiento de chequeo de instancia en el marco DL tradicional. El *chequeo de instancia* consiste en determinar si una aserción se sigue de una Abox. Por ejemplo, $T \cup A \models C(a)$ indica que el individuo a pertenece al concepto C con respecto a la Abox A y Tbox T. Sin embargo, cuando las ontologías son inconsistentes, como las DL son un subconjunto de la Lógica de Primer Orden, la inferencia tiene un efecto explosivo en el sentido de que permite deducir cualquier conclusión. En el marco del chequeo de instancia, esta situación equivale a decir que cuando una ontología es inconsistente, entonces es posible probar que cualquier individuo pertenece a cualquier clase. Está claro que esta situación es altamente indeseable en el marco de cualquier sistema de representación de conocimiento. La argumentación nos brinda un mecanismo apropiado para paliar este problema, permitiendo analizar el chequeo de instancia a través de diferentes nociones alternativas que definiremos a continuación.

Definición 4.5.3 (Pertenencia potencial de un individuo a una clase) *Sea* $\Sigma = (T_S, T_D, A)$ *una δ-ontología. Sea* C *un símbolo de clase definido en* Σ*. Sea* a *un nombre de individuo definido en* Σ*. Sea* $\mathcal{P} = (\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$ *el programa DeLP que se obtiene al traducir la ontología* Σ *mediante la aplicación de la función* $\mathfrak{T}$*. Un individuo* a *pertenece potencialmente una clase* C*, notado como* C_p^a*, si y sólo si existe un argumento* $\mathcal{A}$ *a favor del literal* $C(a)$ *con respecto al programa* $\mathcal{P}$*.*

Definición 4.5.4 (Pertenencia justificada de un individuo a una clase) *Sea* $\Sigma = (T_S, T_D, A)$ *una δ-ontología. Sea* C *un símbolo de clase definido en* Σ*. Sea* a *un nombre de individuo definido en* Σ*. Sea* $\mathcal{P} = (\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$ *el programa rebatible que se obtiene al traducir la ontología* Σ *a DeLP. Un individuo* a *pertenece justificadamente a una clase* C*, notado como* C_j^a*, si y sólo si existe un argumento garantizado a favor del literal* $C(a)$ *con respecto al programa* $\mathcal{P}$*.*

Definición 4.5.5 (Pertenencia estricta de un individuo a una clase) *Sea* $\Sigma = (T_S, T_D, A)$ *una δ-ontología. Sea* C *un símbolo de clase definido en* Σ*. Sea* a *un nombre de*

[14]Por *ontologías pobladas* entendemos ontologías que poseen información no sólo acerca de clases sino también sobre individuos; es decir, sus cajas asercionales no son conjuntos vacíos.

individuo definido en Σ. *Sea* $\mathcal{P} = (\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$ *el programa rebatible que se obtiene al traducir la ontología* Σ *a DeLP. Un individuo a pertenece estrictamente a una clase C, notado como* C_s^a, *si y sólo si existe un argumento vacío a favor del literal* $C(a)$ *con respecto al programa* $\mathcal{P}$.

Definición 4.5.6 (Pertenencia indeterminada de un individuo a una clase) *Sea* $\Sigma = (T_S, T_D, A)$ *una δ-ontología. Sea C un símbolo de clase definido en* Σ. *Sea a un nombre de individuo definido en* Σ. *Sea* $\mathcal{P} = (\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$ *el programa rebatible que se obtiene al traducir la ontología* Σ *a DeLP. La pertenencia de un individuo a a una clase C está indeterminada si no existe ningún argumento a favor del literal* $C(a)$ *con respecto al programa* $\mathcal{P}$.

Hasta aquí, hemos definido la pertenencia de individuos a clases simples, esto es, a clases identificadas por un nombre. Ahora, extenderemos la noción de chequeo de instancia para conceptos arbitrarios.

Definición 4.5.7 (Pertenencias potencial y justificada de un individuo a una clase (versión extendida)) *Sea* $\Sigma = (T_S, T_D, A)$ *una δ-ontología. Sean* C, D *símbolos de clase definidos en* Σ. *Sean* a, b *nombres de individuo definidos en* Σ. *Sea R un nombre de rol atómico. Sea* $\mathcal{P} = (\Pi, \Delta)$ *el programa rebatible que se obtiene al traducir la ontología* Σ *a DeLP tal que:*

$$\begin{array}{rcl} \Pi & = & \mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \text{ y,} \\ \Delta & = & \mathfrak{T}_\Delta(T_D). \end{array}$$

Las pertenencias potencial y justificadas de un individuo a a un concepto complejo se definen como sigue:

- $\neg C$: *El individuo a pertenece potencialmente al concepto* $\neg C$, *notado como* $(\neg C)_p^a$, *si y sólo si existe un argumento* $\langle \mathcal{A}, \sim C(a) \rangle$ *con respecto al programa* $\mathcal{P}$. *Respectivamente, el individuo a pertenece justificadamente al concepto* $\neg C$, *notado como* $(\neg C)_j^a$, *si y sólo si existe un argumento* $\langle \mathcal{A}, \sim C(a) \rangle$ *garantizado con respecto al programa* $\mathcal{P}$.

- $C \sqcap D$: *Sea E un nombre de concepto no presente en* Σ. *Sea* $\mathcal{P}_2$ *el programa DeLP que se obtiene al agregar la traducción de* $C \sqcap D \sqsubseteq E$ *al conjunto de información rebatible,* i.e.*:*

  $$\mathcal{P}_2 = \left(\Pi, \Delta \cup \mathfrak{T}_\Delta(C \sqcap D \sqsubseteq E) \right).$$

  El individuo a pertenece potencialmente al concepto $C \sqcap D$, *notado como* $(C \sqcap D)_p^a$, *si y sólo si existe un argumento* $\langle \mathcal{A}, E(a) \rangle$ *con respecto al programa* $\mathcal{P}_2$. *Respectivamente, el individuo a pertenece justificadamente al concepto* $C \sqcap D$, *notado como*

$(C \sqcap D)^a_j$, *si y sólo si existe un argumento* $\langle \mathcal{A}, E(a) \rangle$ *garantizado con respecto al programa* $\mathcal{P}_2$.

- $C \sqcup D$: *Sea* E *un nombre de concepto no presente en* Σ. *Sea* $\mathcal{P}_2$ *el programa DeLP que se obtiene al agregar la traducción de* $C \sqcup D \sqsubseteq E$ *al conjunto de información rebatible,* i.e.*:*

$$\mathcal{P}_2 = \left(\ \Pi, \Delta \cup \mathfrak{T}_\Delta(C \sqcup D \sqsubseteq E) \ \right).$$

El individuo a *pertenece potencialmente al concepto* $C \sqcup D$, *notado como* $(C \sqcup D)^a_p$, *si y sólo si existe un argumento* $\langle \mathcal{A}, E(a) \rangle$ *con respecto al programa* $\mathcal{P}_2$. *Respectivamente, el individuo* a *pertenece justificadamente al concepto* $C \sqcup D$, *notado como* $(C \sqcup D)^a_j$, *si y sólo si existe un argumento* $\langle \mathcal{A}, E(a) \rangle$ *garantizado con respecto al programa* $\mathcal{P}_2$.

- $\exists R.C$: *Sea* E *un nombre de concepto no presente en* Σ. *Sea* $\mathcal{P}_2$ *el programa DeLP que se obtiene al agregar la traducción de* $\exists R.C \sqsubseteq E$ *al conjunto de información rebatible,* i.e.*:*

$$\mathcal{P}_2 = \left(\ \Pi, \Delta \cup \mathfrak{T}_\Delta(\exists R.C \sqsubseteq E) \ \right).$$

El individuo a *pertenece potencialmente al concepto* $\exists R.C$, *notado como* $(\exists R.C)^a_p$, *si y sólo si existe un argumento* $\langle \mathcal{A}, E(a) \rangle$ *con respecto al programa* $\mathcal{P}_2$. *Respectivamente, el individuo* a *pertenece justificadamente al concepto* $\exists R.C$, *notado como* $(\exists R.C)^a_j$, *si y sólo si existe un argumento* $\langle \mathcal{A}, E(a) \rangle$ *garantizado con respecto al programa* $\mathcal{P}_2$.

Nótese que el concepto E en realidad puede ser más grande que los conceptos complejos que se pretenden analizar pues la expresión E solamente es introducida con el objeto de poder expresar la consulta correspondiente. Lo mismo ocurre en la noción extendida de de pertenencia estricta, que se da a continuación.

Definición 4.5.8 (Pertenencia estricta de un individuo a una clase (versión extendida)) *Sea* $\Sigma = (T_S, T_D, A)$ *una* δ*-ontología. Sean* C, D *símbolos de clase definidos en* Σ. *Sean* a, b *nombres de individuo definidos en* Σ. *Sea* R *un nombre de rol atómico. Sea* $\mathcal{P} = (\Pi, \Delta)$ *el programa rebatible que se obtiene al traducir la ontología* Σ *a DeLP tal que:*

$$\begin{array}{rcl} \Pi & = & \mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \text{ y,} \\ \Delta & = & \mathfrak{T}_\Delta(T_D). \end{array}$$

La pertenencia estricta de un individuo a *a un concepto complejo se define como sigue:*

- $\neg C$: *El individuo a pertenece estrictamente al concepto* $\neg C$*, notado como* $(\neg C)^a_s$*, si y sólo si existe un argumento* $\langle \emptyset, \sim C(a) \rangle$ *con respecto al programa* $\mathcal{P}$*.*

- $C \sqcap D$: *Sea* E *un nombre de concepto no presente en* Σ*. Sea* $\mathcal{P}_2$ *el programa DeLP que se obtiene al agregar la traducción de* $C \sqcap D \sqsubseteq E$ *al conjunto de información estricta,* i.e.*:*

$$\mathcal{P}_2 = \left(\ \Pi \cup \mathfrak{T}_\Pi(C \sqcap D \sqsubseteq E), \mathfrak{T}_\Delta(T_D) \ \right).$$

  El individuo a pertenece potencialmente al concepto $C \sqcap D$*, notado como* $(C \sqcap D)^a_s$*, si y sólo si existe un argumento* $\langle \emptyset, E(a) \rangle$ *con respecto al programa* $\mathcal{P}_2$*.*

- $C \sqcup D$: *Sea* E *un nombre de concepto no presente en* Σ*. Sea* $\mathcal{P}_2$ *el programa DeLP que se obtiene al agregar la traducción de* $C \sqcup D \sqsubseteq E$ *al conjunto de información estricta,* i.e.*:*

$$\mathcal{P}_2 = \left(\ \Pi \cup \mathfrak{T}_\Pi(C \sqcup D \sqsubseteq E), \mathfrak{T}_\Delta(T_D) \ \right).$$

  El individuo a pertenece estrictamente al concepto $C \sqcup D$*, notado como* $(C \sqcup D)^a_s$*, si y sólo si existe un argumento* $\langle \emptyset, E(a) \rangle$ *con respecto al programa* $\mathcal{P}_2$*.*

- $\exists R.C$: *Sea* E *un nombre de concepto no presente en* Σ*. Sea* $\mathcal{P}_2$ *el programa DeLP que se obtiene al agregar la traducción de* $\exists R.C \sqsubseteq E$ *al conjunto de información estricta,* i.e.*:*

$$\mathcal{P}_2 = \left(\ \Pi \cup \mathfrak{T}_\Pi(\exists R.C \sqsubseteq E), \mathfrak{T}_\Delta(T_D) \ \right).$$

  El individuo a pertenece estrictamente al concepto $\exists R.C$*, notado como* $(\exists R.C)^a_s$*, si y sólo si existe un argumento* $\langle \emptyset, E(a) \rangle$ *con respecto al programa* $\mathcal{P}_2$*.*

Ejemplo 4.5.2 (Continúa el Ejemplo 4.4.10) *Considere nuevamente la* δ*-ontología* $\Sigma_{4.4.1}$ *(página 115) cuya semántica está dada por el programa DeLP* $\mathcal{P}_{4.4.1} = (\Pi, \Delta)$ *(página 120). En dicho programa DeLP, el individuo "Tweety" pertenece estrictamente al concepto "pingüino" en forma trivial pues el literal* $penguin(tweety)$ *pertenece a* Π*. Además, el individuo "Tina" pertenece estrictamente al concepto "ave" pues el literal* $bird(tina)$ *se deriva estrictamente a partir de* $\mathcal{P}_{4.4.1}$*. Tina pertenece potencialmente tanto al concepto "vuela" como a su complemento ya que existen argumentos a favor de "*$flies(tina)$*" como de "*$\sim flies(tina)$*", a saber:*

$$\begin{array}{rcl} \mathcal{A}_1 & = & \left\{ \ flies(tina) \prec chicken(tina), scared(tina) \ \right\}, y \\ \mathcal{A}_2 & = & \left\{ \ \sim flies(tina) \prec chicken(tina) \ \right\}. \end{array}$$

El individuo "Tina" pertenece justificadamente al concepto "vuela" porque existe un argumento garantizado a favor del literal "flies(tina)" , a saber, el argumento $\mathcal{A}_1$. Es de notar que como el argumento $\mathcal{A}_1$ es estrictamente más específico que $\mathcal{A}_2$, no puede atacar a este último durante la construcción del árbol dialéctico en la resolución de la consulta (García and Simari, 2004, Ejemplo 3.5).

Ejemplo 4.5.3 (Continúa el Ejemplo 4.4.11) *Consideremos nuevamente la δ-ontologia $\Sigma_{4.4.2}$ (página 116), cuya semántica es el programa DeLP $\mathcal{P}_{4.4.2}$ (página 120). En este caso el individuo "Nixon" pertenece estrictamente al concepto "habitante de Chicago" pues el "lives_in_chicago(nixon) pertenece a Π. En cambio, el individuo "Nixon" pertenece potencialmente tanto al concepto "armado" porque existen argumentos tanto a favor del literal "has_a_gun(nixon)" como del literal "$\sim$has_a_gun(nixon)", a saber:*

$$\begin{array}{rcl} \mathcal{A}_1 & = & \left\{ \; has\_a\_gun(nixon) \prec lives\_in\_chicago(nixon) \; \right\}, y \\ \mathcal{A}_2 & = & \left\{ \begin{array}{l} \sim has\_a\_gun(nixon) \prec lives\_in\_chicago(nixon), pacifist(nixon) \\ pacifist(nixon) \prec quaker(nixon) \end{array} \right\}. \end{array}$$

Además, el individuo "Nixon" pertenece justificadamente al concepto "armado" pues existe un argumento garantizado para el literal "has_a_gun(nixon)" (ver la Figura 4.4.(a)). Existen dos argumentos adicionales $\langle \mathcal{A}_3, \sim pacifist(nixon) \rangle$ y $\langle \mathcal{A}_4, pacifist(nixon) \rangle$, con:

$$\begin{array}{rcl} \mathcal{A}_3 & = & \left\{ \; \sim pacifist(nixon) \prec republican(nixon) \; \right\}, y \\ \mathcal{A}_4 & = & \left\{ \; pacifist(nixon) \prec quaker(nixon) \; \right\}. \end{array}$$

*El argumento $\mathcal{A}_1$ es atacado propiamente por $\mathcal{A}_2$, quien, a su vez, es atacado por bloqueo, por el argumento $\mathcal{A}_3$ en el subargumento $\mathcal{A}_4$. Sin embargo, no es posible decidir si el individuo "Nixon" pertenece o no en forma justificada al concepto "pacifista" pues el resultado para la consulta "pacifist(nixon)" es "*INDECISO*". Esto ocurre porque los argumentos $\mathcal{A}_3$ y $\mathcal{A}_4$ se derrotan mutuamente por bloqueo (García and Simari, 2004) (ver la Figura 4.4.(b)-(c)).*

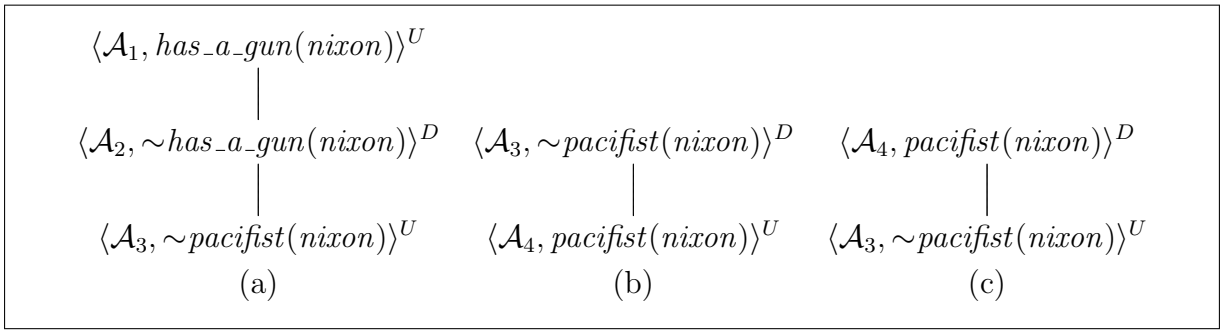


Figura 4.4: Análisis dialéctico para el programa $\mathcal{P}_{4.4.2}$

Ejemplo 4.5.4 (Continúa el Ejemplo 4.4.12) *Consideremos nuevamente la δ-ontología $\Sigma_{4.4.3}$ (página 116), la cual tiene como semántica al programa DeLP $\mathcal{P}_{4.4.3}$ (página 121). En esta ontología, el individuo "Acme" pertenece en forma potencial al "$\neg buy\_stock$" pues existe un argumento a favor del literal "$\sim buy\_stock(acme)$" con respecto a $\mathcal{P}_{4.4.3}$:*

$$\mathcal{A} = \left\{ \begin{array}{l} \sim buy\_stock(acme) \prec good\_price(acme), risky\_company(acme) \\ risky\_company(acme) \prec in\_fusion(acme, steel) \end{array} \right\}.$$

Además, en este caso el individuo "Acme" pertenece en forma justificada al concepto "Comprar accciones" pues existe un argumento garantizado para el literal "$buy\_stock(acme)$" respecto de $\mathcal{P}_{4.4.3}$. Consideremos los argumentos:

$$\begin{array}{rcl} \mathcal{A}_1 & = & \left\{ buy\_stock(acme) \prec good\_price(acme) \right\}; \\ \mathcal{A}_2 & = & \left\{ \begin{array}{l} \sim buy\_stock(acme) \prec good\_price(acme), risky\_company(acme) \\ risky\_company(acme) \prec in\_fusion(acme, steel) \end{array} \right\}, y \\ \mathcal{A}_3 & = & \left\{ \sim risky\_company(acme) \prec in\_fusion(acme, steel), strong(steel) \right\}. \end{array}$$

El argumento $\mathcal{A}_1$ es derrotado por $\mathcal{A}_2$, el cual a su vez es derrotado por $\mathcal{A}_3$ (García and Simari, 2004).

Ejemplo 4.5.5 (Continúa el Ejemplo 4.4.13) *Considere nuevamente la δ-ontología $\Sigma_{4.4.4}$ (página 117), cuya semántica está dada por el programa DeLP $\mathcal{P}_{4.4.4}$ (página 121). Aquí, el individuo "Clyde" pertenece en forma justificada al concepto "$\neg gray$" pues existe un argumento garantizado $\mathcal{A}$ a favor del literal "$\sim gray(clyde)$":*

$$\mathcal{A} = \left\{ \sim gray(clyde) \prec royal\_elephant(clyde) \right\}.$$

Ejemplo 4.5.6 (Continúa el Ejemplo 4.4.14) *Veamos nuevamente la ontología $\Sigma_{4.4.5}$ (página 117) cuyo significado está dado por el programa DeLP $\mathcal{P}_{4.4.5} = (\Pi, \Delta)$ (página 121). En esta ontología "ken" pertenece estrictamente al concepto "adulto" ($adult(ken) \in \Pi$); "ken" pertenece potencialmente a los conceptos "worker" y "$\neg worker$" pues existen argumentos a favor de los literales "$worker(ken)$" y "$\sim worker(ken)$", respectivamente. En cambio, sólo se puede decir que "ken" pertenece justificadamente al concepto $\neg worker$ pues solamente existe un argumento garantizado para el literal "$\sim worker(ken)$" (Simari and Loui, 1992, Ejemplo 6.5).*

Ejemplo 4.5.7 (Continúa el Ejemplo 4.4.15) *Considere nuevamente la δ-ontología $\Sigma_{4.4.6}$ interpretada como el programa DeLP $\mathcal{P}_{4.4.6}$. Para determinar si "John" pertenece la clase de los "violentos", es necesario determinar si existe un argumento garantizado a favor del literal "$violent(john)$".*

Así, encontramos un argumento $\langle \mathcal{A}_1, violent(john) \rangle$ a favor de que John es violento, donde:

$$\mathcal{A}_1 = \left\{ violent(john) \prec boxer(john) \right\}.$$

Pero hay otro argumento que derrota al primero, indicando que John no es violento: $\langle \mathcal{A}_2, \sim violent(john) \rangle$*, con:*

$$\mathcal{A}_2 = \left\{ \begin{array}{l} \sim violent(john) \prec boxer(john), animal\_lover(john) \\ animal\_lover(john) \prec loves(john, lola), airedale(lola) \end{array} \right\}.$$

Finalmente, aparece un tercer argumento $\langle \mathcal{A}_3, \sim animal\_lover(john) \rangle$*, socavando la conclusión anterior en el punto de ataque* $animal\_lover(john)$*, derrotando al segundo por bloqueo, y reinstaurando al primero (ver la Figura 4.5):*

$$\mathcal{A}_3 = \left\{ \sim animal\_lover(john) \prec kicks(john, tina), poodle(tina) \right\}.$$

Así, como el resultado a las consultas $violent(john)$ *es* Si *y* $\sim violent(john)$ *es* No*, concluimos que John pertenece a la clase de los violentos.*

$$\langle \mathcal{A}_1, violent(john) \rangle^U$$
$$|$$
$$\langle \mathcal{A}_2, \sim violent(john) \rangle^D$$
$$|$$
$$\langle \mathcal{A}_3, \sim animal\_lover(john) \rangle^U$$

Figura 4.5: Árbol dialéctico para la consulta $violent(john)$ respecto del programa DeLP $\mathcal{P}_{4.4.6}$

Ejemplo 4.5.8 (Continúa el Ejemplo 4.4.16) *Considere nuevamente la* δ*-ontología* $\Sigma_{4.4.7} = (\emptyset, T_D, A)$ *(página 118) interpretada como el programa DeLP* $\mathcal{P}_{4.4.7} = (\Pi, \Delta)$ *(página 122). Si utilizamos como criterio de comparación de argumentos aquel definido en términos de comparación de reglas (ver la Sección 3.3.4 en la página 77), obtenemos el siguiente análisis dialéctico al momento de considerar las pertenencias justificadas de distintos individuos a las diversas clases definidas en la ontología.*

Mostraremos los argumentos que surgen a partir de $\mathcal{P}_{4.4.7}$ *y caracterizaremos sus interacciones en el análisis dialéctico que surge cuando los consideramos. Existe un argumento* $\mathcal{A}_1$ *soportando la conclusión rebatible que indica que Avenger vuela,* i.e. $\langle \mathcal{A}_1, fly(avenger) \rangle$ *donde:*

$$\mathcal{A}_1 = \left\{ \begin{array}{l} fly(avenger) \prec bird(avenger) \\ bird(avenger) \prec eagle(avenger) \end{array} \right\}.$$

Por lo tanto, se puede decir que el individuo "Avenger" pertenece potencialmente al concepto "volador".

Asumiendo que el criterio de comparación establece que $r_8 \succ r_2$*, este argumento es derrotado por un argumento* $\langle \mathcal{A}_2, \sim fly(avenger) \rangle$ *soportando que "Avenger" pertenece potencialmente al complemento de la clase* fly *porque tiene un ala rota, donde:*

$$\mathcal{A}_2 = \left\{ \sim fly(avenger) \prec eagle(avenger), hasBrokenWing(avenger) \right\}.$$

Como el argumento $\mathcal{A}_2$ no tiene derrotadores, el argumento $\mathcal{A}_1$ está por lo tanto derrotado y es marcado como un nodo-D. Así, concluimos que "Avenger" pertenece justificadamente a la clase $\neg fly$. El correspondiente árbol de dialéctica es mostrado en la Figura 4.6.(a).

De la misma manera, existe un argumento $\mathcal{B}_1$ soportando la conclusión rebatible que dice que Opus vuela (u "Opus" pertenece potencialmente al concepto "fly"), i.e., *$\langle \mathcal{B}_1, fly(opus) \rangle$ donde:*

$$\mathcal{B}_1 = \left\{ \begin{array}{l} fly(opus) \prec bird(opus) \\ bird(opus) \prec penguin(opus) \end{array} \right\}.$$

Otro argumento $\langle \mathcal{B}_2, \sim fly(opus) \rangle$ puede ser construido, soportando la conclusión que indica que el individuo "Opus" pertenece potencialmente a la clase "$\neg$fly", con:

$$\mathcal{B}_2 = \left\{ \sim fly(opus) \prec penguin(opus) \right\}.$$

El argumento $\mathcal{B}_2$ derrota $\mathcal{B}_1$ bajo la hipótesis que el criterio de comparación de reglas establece que $r_5 \succ r_2$. Sin embargo, bajo la suposición que $r_7 \succ r_5$, el argumento $\mathcal{B}_2$ es derrotado por otro argumento $\langle \mathcal{B}_3, fly(opus) \rangle$ que reinstaura al argumento $\mathcal{B}_1$, donde:

$$\mathcal{B}_3 = \left\{ \begin{array}{l} fly(opus) \prec geneticallyAlteredPenguin(opus) \\ geneticallyAlteredPenguin(opus) \prec \\ \qquad penguin(opus), \\ \qquad isOperatedBy(opus, frankenstein), \\ \qquad geneticSurgeon(frankenstein) \\ isOperatedBy(opus, frankenstein) \prec operates(frankenstein, opus) \\ geneticSurgeon(frankenstein) \prec \\ \qquad geneticist(frankenstein), \\ \qquad surgeon(frankenstein) \end{array} \right\}.$$

Por lo tanto, el árbol asociado al literal "fly(opus)" tiene tres nodos, con su raíz etiquetada como nodo-U (ver la Figura 4.6.(b)). Entonces, el argumento para "fly(opus)" está garantizado y, en consecuencia, el individuo "Opus" pertenece potencialmente a la clase "fly".

Existen dos argumentos más $\langle \mathcal{C}_1, animal(avenger) \rangle$ y $\langle \mathcal{D}_1, animal(opus) \rangle$, soportando las conclusiones rebatibles que Avenger y Opus son potencialmente animales respectivamente, donde:

$$\mathcal{C}_1 = \left\{ \begin{array}{l} animal(avenger) \prec bird(avenger) \\ bird(avenger) \prec eagle(avenger) \end{array} \right\}, \text{ y}$$

$$\mathcal{D}_1 = \left\{ \begin{array}{l} animal(opus) \prec bird(opus) \\ bird(penguin) \prec penguin(opus) \end{array} \right\}.$$

Estos dos argumentos no tienen derrotadores, ellos están por lo tanto garantizados y los árboles dialécticos tendrán un único nodo, como se muestra en la Figura 4.6.(c–d). Así, "Avenger" y "Opus" pertenecen en forma justificada al concepto "animal".

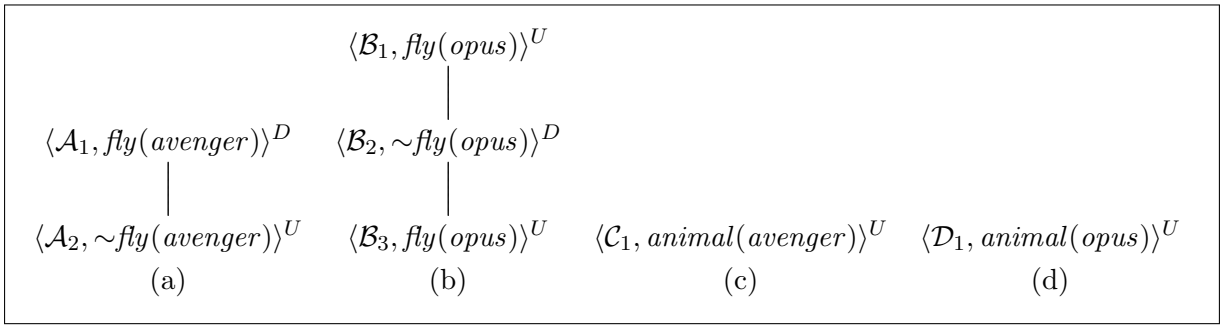


Figura 4.6: Árboles dialécticos para las consultas: (a) *fly*(*avenger*), (b) *fly*(*opus*), (c) *animal*(*avenger*), y (d) *animal*(*opus*)

Ejemplo 4.5.9 (Continúa el Ejemplo 4.4.17) *Consideremos el programa DeLP presentado en el Ejemplo 4.4.17 (página 123). El individuo "Mary" pertenece en forma justificada al concepto "pass" porque existe un argumento garantizado a favor del literal "pass(mary)"; en cambio, el individuo "Paul" sólo pertenece potencialmente al concepto "pass" (véase el Ejemplo 3.3.20 para la justificación de ambas afirmaciones).*

Ejemplo 4.5.10 (Continúa el Ejemplo 2.4.4) *Consideremos nuevamente en el Ejemplo 2.4.4 presentado en la página 49. Si consideramos a dicha ontología poblada con dos individuos, digamos un individuo llamado "Flor" perteneciente al concepto "marriedWoman" y otro llamado "Leticia" perteneciente al concepto "divorcee". Cuando introducimos dicha ontología en Racer obtenemos la salida que indica que la Abox es incoherente. Consideremos la siguiente δ-ontología para modelar la situación descripta:*

$$T_S = \left\{ \begin{array}{l} marriedWoman \sqsubseteq woman \\ divorcee \sqsubseteq hadHusband \end{array} \right\}$$

$$T_D = \left\{ \begin{array}{l} marriedWoman \sqsubseteq \neg divorcee \\ hadHusband \sqsubseteq \neg hasHusband \\ hasHusband \sqsubseteq marriedWoman \\ hadHusband \sqsubseteq marriedWoman \end{array} \right\}$$

$$A = \left\{ \begin{array}{l} flor : marriedWoman \\ leticia : divorcee \end{array} \right\}.$$

Esta δ-ontología es interpretada como el programa DeLP (Π, Δ) *con:*

$$\Pi = \left\{ \begin{array}{l} woman(X) \leftarrow marriedWoman(X) \\ \sim marriedWoman(X) \leftarrow \sim woman(X) \\ hadHusband(X) \leftarrow divorcee(X) \\ \sim divorcee(X) \leftarrow \sim hadHusband(X) \\ marriedWoman(flor) \\ divorcee(leticia) \end{array} \right\}, y$$

$$\Delta = \left\{ \begin{array}{l} \sim divorcee(X) \prec marriedWoman(X) \\ \sim hasHusband(X) \prec hadHusband(X) \\ marriedWoman(X) \prec hasHusband(X) \\ marriedWoman(X) \prec hadHusband(X) \end{array} \right\}.$$

Al considerar la pertenencia del individuo "Flor" a los distintos conceptos obtenemos las siguientes respuestas: "Flor" pertenece en forma estricta al concepto "woman"; esto es así porque existe un argumento $\langle \emptyset, woman(flor) \rangle$ *cuya derivación se basa en la regla estricta:*

$$woman(flor) \leftarrow marriedWoman(flor).$$

Trivialmente, "Flor" pertenece en forma estricta al concepto "marriedWoman", pues el literal $marriedWoman(flor)$ *pertenece a* Π*. "Flor" pertenece en forma garantizada al complemento del concepto "divorcee" pues existe un argumento garantizado* $\langle \mathcal{A}, \sim divorcee(flor) \rangle$ *con:*

$$\mathcal{A} = \{ \sim divorcee(flor) \prec marriedWoman(flor) \}$$

*cuyo árbol de dialéctica posee un único nodo. En cambio, no es posible determinar la pertenencia garantizada de "Flor" al concepto "hadHusband" pues el resultado de la consulta "hadHusband(flor)" tiene como respuesta "*Indeciso*"; esto ocurre porque no es posible construir argumentos a favor de tal literal. Lo misma situación se da para el concepto "hasHusband" con respecto al individuo "Flor".*

En cambio, al considerar la pertenencia del individuo "Leticia" a los distintos conceptos, obtenemos los siguientes resultados: La respuesta para la consulta $woman(leticia)$ *es* Si*, pues existe un argumento garantizado (ya que no posee derrotadores)* $\langle \mathcal{B}, woman(leticia) \rangle$ *basado en la derivación que se muestra en la Figura 4.7. La respuesta para la consulta* $\sim divorcee(leticia)$ *es* No *pues* $divorcee(leticia)$ *pertenece al conjunto de hechos del programa* (Π, Δ)*. La respuesta para la consulta* $hadHusband(leticia)$ *es* Si*, pues este literal se puede derivar a partir de un subargumento del argumento* $\mathcal{A}$ *y no tiene derrotadores. La respuesta para la consulta* $hasHusband(leticia)$ *es* No*, pues existe un argumento* $\mathcal{C}$ *garantizado a favor del literal* $\sim hasHusband(leticia)$ *(ver la Figura 4.8). La respuesta para la consulta* $marriedWoman(leticia)$ *es* Si*, pues existe un argumento garantizado* $\mathcal{D}$ *a favor de dicho literal (véase la Figura 4.9).*

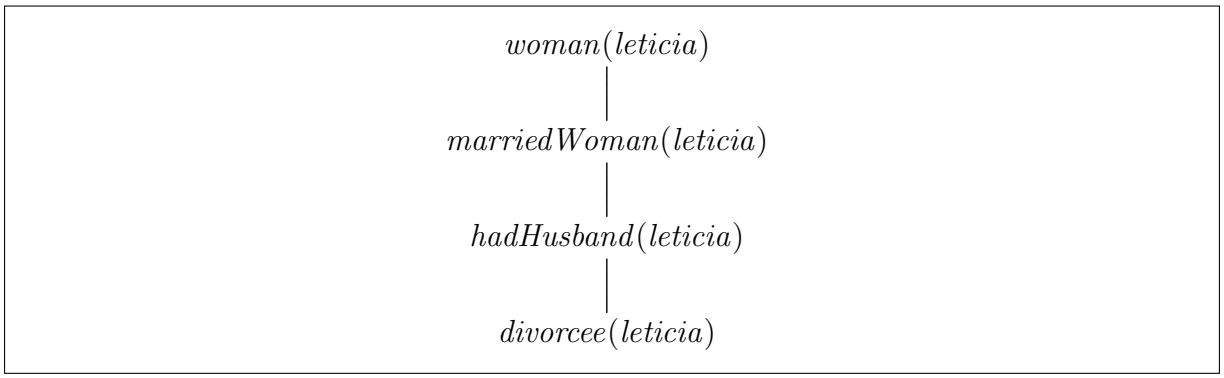


Figura 4.7: Estructura de argumento $\langle \mathcal{B}, woman(leticia) \rangle$

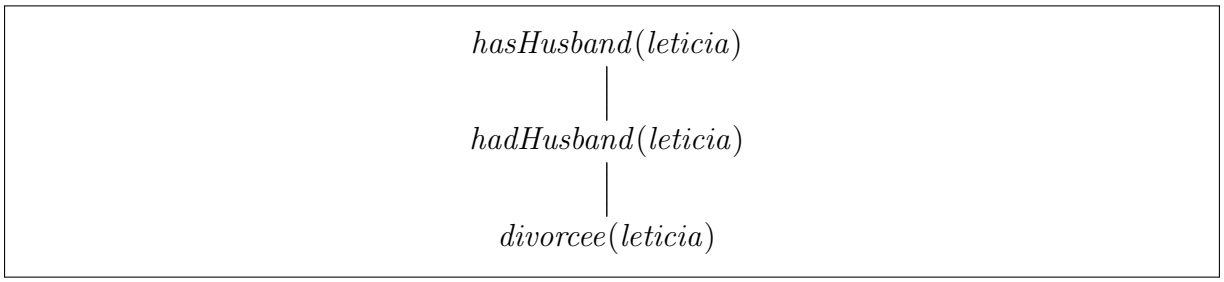


Figura 4.8: Estructura de argumento $\langle \mathcal{C}, hasHusband(leticia) \rangle$

Ejemplo 4.5.11 (Continúa el Ejemplo 4.4.18) *En el contexto del programa DeLP $\mathcal{P}_{4.4.18}$, no es posible decidir la pertenencia del individuo x al concepto D pero sí es posible decidir que x debe pertenecer al concepto C como mostramos a continuación.*

En el caso del concepto D, vemos que tenemos dos $\langle \mathcal{A}, d(x) \rangle$ y $\langle \mathcal{B}, \sim d(x) \rangle$ argumentos a favor y en contra de la pertenencia potencial de x al concepto D, con:

$$\begin{array}{rcl} \mathcal{A} & = & \left\{ \; d(x) \prec a(x) \; \right\} \; y \\ \mathcal{B} & = & \left\{ \begin{array}{l} \sim d(x) \prec b(x) \\ b(x) \prec a(x) \end{array} \right\}. \end{array}$$

Como estos argumentos se atacan por bloqueo mutuamente, la respuesta a la consulta $d(x)$ es Indeciso.

Sin embargo, al analizar la pertenencia de x al concepto C, vemos que tenemos un único argumento a favor de la pertenencia potencial de x a C; a saber, $\langle \{c(x) \prec a(x)\}, c(x) \rangle$. Por lo tanto, este argumento se halla garantizado y es posible afirmar que x pertenece en forma justificada al concepto C.

4.5.2. Recuperación (*Retrieval*)

En la Sección 2.3.2 (página 36), presentamos el problema de la *recuperación* para el marco DL tradicional. Si consideramos a una base de conocimiento como un medio

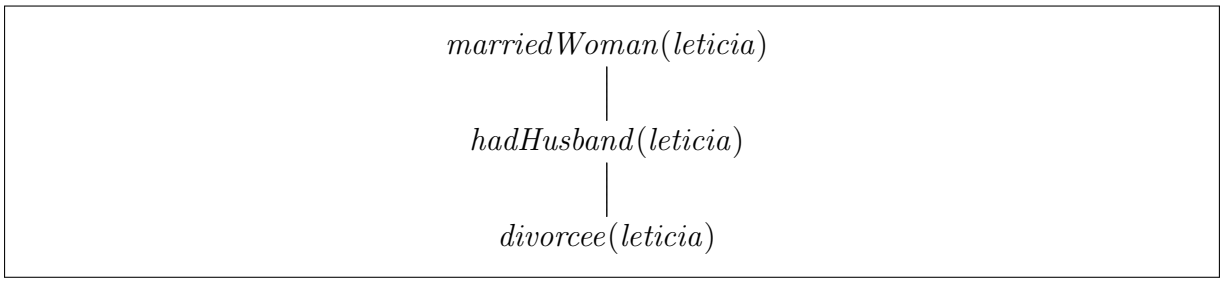


Figura 4.9: Estructura de argumento $\langle \mathcal{D}, marriedWoman(leticia) \rangle$

para almacenar información acerca de individuos, podríamos querer conocer todos los individuos que son instancias de una descripción de concepto C. Dada una ontología (T, A) y un concepto C, en el *problema de la recuperación* nos interesa conocer todos los individuos a tales que $T \cup A \models a : C$. Una versión no optimizada para el problema de la recuperación puede realizarse verificando para cada individuo de la Abox A si es una instancia del concepto C.

En el marco de nuestro acercamiento, resolvemos dos subproblemas que son:

- *Recuperación abierta:* Dada una δ-ontología $\Sigma = (T_S, T_D, A)$ y una clase C, nos interesa hallar todos los individuos a que son instancias de C. El enfoque usado para resolver este problema es el *naïve*. Resolvemos este problema hallando a todos los individuos a tal que existe un argumento garantizado $\langle \mathcal{A}, C(a) \rangle$ con respecto al programa DeLP $(\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(T_S), \mathfrak{T}_\Delta(T_D))$, que hace las veces de interpretación de Σ. La versión algorítmica de tal acercamiento se presenta en la Figura 4.10.

- *Recuperación de todas las clases:* Dado un individuo a, determinar todas las clases (con nombre) en Σ de las que a es una instancia. Resolvemos este problema encontrando todas las clases C tales que existe un argumento garantizado $\langle \mathcal{A}, C(a) \rangle$ con respecto al programa DeLP $(\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(T_S), \mathfrak{T}_\Delta(T_D))$, que hace las veces de interpretación de Σ. El algoritmo para realizar tal inferencia se muestra en la Figura 4.11.

Debido que los árboles de dialéctica de la Programación en Lógica Rebatible son finitos, esto garantizará que el tiempo de ejecución para estas tareas de razonamiento también lo es (ver la Propiedad 5.1.18 en la página 202).

4.5.3. Consistencia de terminologías

Dada una δ-ontología $\Sigma = (T_S, T_D, \emptyset)$, *i.e.* sin aserciones, el programa DeLP resultante de la aplicación de la función de interpretación semántica $\mathfrak{T}$ no contiene hechos. Así, el motor de inferencia DeLP no será capaz de inferir ninguna información respecto de

```
function Retrieval( Σ = (T_S, T_D, A); c : Concept ) : Set of Individual
var S : Set of Individual
begin
        I := Set of individuals appearing in Σ = (T_S, T_D, A)
        S := ∅
        for each individual name a ∈ I do
            if ⟨A, c(a)⟩ is warranted then
                S := S ∪ {a}
            end if
        end for
        return S
end
```

Figura 4.10: Recuperación abierta: Dada una clase C, determinar todos los individuos a que son instancias de C

```
function RetrievalOfAllClasses( Σ = (T_S, T_D, A); a : Individual ) : Set of Class
var S : Set of Class
begin
        C := Set of concepts appearing in Σ = (T_S, T_D, A)
        S := ∅
        for each concept name c ∈ C do
            if ⟨A, c(a)⟩ is warranted then
                S := S ∪ {c}
            end if
        end for
        return S
end
```

Figura 4.11: Recuperación de todas las clases: Dado un individuo a, recuperar todas las clases de las que a es una instancia

las clases presentes en la ontología. Aún así, en el caso en el que la terminología fuera insatisfacible, es deseable detectar tal situación en el marco de la programación en lógica.

Una solución a este problema consiste en traducir el conjunto $T_S \cup T_D$ como un programa AnsProlog$^{\neg,\bot}$ (Baral, 2003) enriquecido con información respecto de la insatisfacibilidad de clases. Cada axioma DL en $T_S \cup T_D$ es transformado de acuerdo a la función $\mathfrak{T}_\Pi$. Además, por cada nombre de concepto, una restricción es agregada para prevenir la existencia de literales complementarios.

Ejemplo 4.5.12 *Considérese nuevamente la δ-ontología $\Sigma_{4.4.7} = (\emptyset, T_D, A)$ presentada en el Ejemplo 4.4.7 (página 118). La transformación del conjunto T_D en AnsProlog$^{\neg,\bot}$ resulta en el programa $\mathcal{P}^{LP}_{4.4.7}$ presentado en la Figura 4.12, donde cada sentencia s_i es presentada como la regla AnsProlog a_i.*

$$
\mathcal{P}^{LP}_{4.4.7} = \left\{ \begin{array}{ll}
a_1: & animal(X) \leftarrow bird(X) \\
a_2: & fly(X) \leftarrow bird(X) \\
a_3: & bird(X) \leftarrow eagle(X) \\
a_4: & bird(X) \leftarrow penguin(X) \\
a_5: & \neg fly(X) \leftarrow penguin(X) \\
a_6: & geneticallyAlteredPenguin(X) \leftarrow \\
 & \qquad penguin(X), isOperatedBy(X,Y), geneticSurgeon(Y) \\
 & \ldots \\
a^1_{10}: & geneticSurgeon(X) \leftarrow \\
 & \qquad geneticist(X), surgeon(X) \\
a^2_{10}: & geneticist(X) \leftarrow geneticSurgeon(X) \\
a^3_{10}: & surgeon(X) \leftarrow geneticSurgeon(X) \\
 & \bot \leftarrow animal(X), \neg animal(X) \\
 & \bot \leftarrow bird(X), \neg bird(X) \\
 & \bot \leftarrow fly(X), \neg fly(X) \\
 & \bot \leftarrow penguin(X), \neg penguin(X) \\
 & \ldots
\end{array} \right\}
$$

Figura 4.12: Programa AnsProlog$^{\neg,\bot}$ obtenido a partir de la ontología $\Sigma_{4.4.7}$

Ejemplo 4.5.13 (Continúa el Ejemplo 4.4.3) *Consideremos nuevamente la δ-ontología $\Sigma_{4.4.3} = (\emptyset, T_D, A)$, para detectar inconsistencias en la jerarquía de clases derivada de la misma utilizaremos el programa AnsProlog$^{\neg,\bot}$:*

$$\mathcal{P}^{LP}_{4.4.3} = \left\{ \begin{array}{l} buy\_stock(X) \leftarrow good\_price(X) \\ \sim buy\_stock(X) \leftarrow good\_price(X), risky\_company(X) \\ risky\_company(X) \leftarrow in\_fusion(X,Y) \\ risky\_company(X) \leftarrow closing(X) \\ \sim risky\_company(X) \leftarrow in\_fusion(X,Y), strong(Y) \\ \bot \leftarrow buy\_stock(X), \neg buy\_stock(X) \\ \bot \leftarrow good\_price(X), \neg good\_price(X) \\ \bot \leftarrow risky\_company(X), \neg risky\_company(X) \\ \bot \leftarrow strong(X), \neg strong(X) \\ \bot \leftarrow closing(X), \neg closing(X) \end{array} \right\}.$$

4.6. Caso de estudio: Integración de ontologías

La *integración de datos* es el problema de combinar datos residiendo en diferentes fuentes y el de proveer al usuario con una vista unificada de dichos datos (Calvanese et al., 2001; Lenzerini, 2002). El problema de diseñar sistemas de integración de datos es particularmente importante en el contexto de aplicaciones en la Web Semántica donde las ontologías son desarrolladas independientemente unas de otras, y por esta razón pueden ser mutuamente inconsistentes.

Dada una ontología, nos interesa saber si un individuo es una instancia de un cierto concepto. Cuando tenemos varias ontologías, los mismos conceptos pueden tener nombres distintos para un mismo significado o aún nombres iguales para significados diferentes. Entonces, para relacionar los conceptos entre dos ontologías diferentes se utilizan *reglas puente* (también conocidas como reglas de articulación (Mitra, 2004; Mitra et al., 2000; Mitra and Wiederhold, 2001, 2002)). De esta manera, un concepto corresponde a una *vista* sobre otros conceptos de otra ontología.

Hay dos variaciones al problema de contestar consultas usando sistemas de información basados en ontologías: *integración de arriba hacia abajo* (conocida también *top-down* o *global-as-view* e *integración de abajo hacia arriba* (conocida también como *bottom-up* o *local-as-view*) (para más detalles, véase la Sección 2.5). En el enfoque de arriba hacia abajo, los conceptos usados en las consultas son obtenidos a partir de una ontología global. La ontología global contiene conceptos obtenidos a partir de las ontologías fuente. En el acercamiento de abajo hacia arriba, los términos usados en la ontología global son diferentes a los utilizados en las ontologías fuente. Los términos usados en las ontologías fuente están definidos en base a términos definidos en la ontología global.

En el caso de estudio que presentaremos a continuación, mostraremos cómo razonar con sistemas de integración de ontologías en ambos acercamientos en el caso en que las ontologías posean inconsistencias.

4.6.1. Integración de ontologías de arriba hacia abajo

Una posible arquitectura para los sistemas de integración está basada en un esquema global y un conjunto de fuentes locales. Las fuentes locales contienen los datos reales, mientras que el esquema global provee una vista reconciliada de las fuentes subyacentes (Lenzerini, 2002). Tal acercamiento se llama *top-down* o *global-as-view*. En el modelo global-as-view, el usuario final especifica una consulta usando los términos de la ontología global. Una *regla puente* relaciona un concepto de una ontología con uno o más conceptos obtenidos de otras ontologías. En nuestra presentación, asumiremos que las ontologías así como las reglas puente están expresadas como ontologías DL, éstas serán interpretadas como programas DeLP. Un servicio básico provisto por los sistemas de integración de datos es el de responder a las consultas realizadas en términos del esquema global. Así, una regla puente relaciona un concepto (representado por la cabeza de una regla) con uno o varios conceptos (representados por el cuerpo de una regla). Así un sistema de integración de ontologías *reescribe* una consulta reemplazando un concepto de la ontología global por uno o varios conceptos de las ontologías fuente. Veremos cómo tal reemplazo servirá para la construcción de argumentos y cómo, en el caso en que los argumentos formados lleven a conclusiones contradictorias, se utilizará un análisis dialéctico para determinar qué conclusiones se hallan finalmente garantizadas.

Definición abstracta del sistema de integración de ontologías

En esta sección, presentamos un marco lógico basado en DeLP para la integración de ontologías. Nuestra presentación es una variación del marco general presentado por Lenzerini (Lenzerini, 2002) pero adaptado a un marco argumentativo. En particular, presentamos un marco basado en un esquema global. Esto es, proponemos un sistema de integración de ontologías cuyo objeto es combinar datos residiendo en fuentes diferentes y el de proveer al usuario con una *vista unificada* de tales datos. Tal vista unificada es representada por el esquema global, el cual puede ser consultado por el usuario. Para relacionar las definiciones de datos presentadas en el esquema global y aquellas de las fuentes locales, se definirá una correspondencia entre el esquema global y las fuentes locales.

Definición 4.6.1 (Sistema abstracto de integración de ontologías) *Un sistema abstracto de integración de ontologías $\mathcal{I}$ es una tripla $(\mathcal{G}, \mathcal{S}, \mathcal{M})$ donde:*

- *$\mathcal{G}$ es una ontología global expresada en un lenguaje $\mathcal{L}_\mathcal{G}$ sobre un alfabeto $\mathcal{A}_\mathcal{G}$. El alfabeto comprende un símbolo para cada alfabeto de $\mathcal{G}$.*
- *$\mathcal{S}$ es un conjunto de n ontologías fuente locales $\mathcal{S}_1, \ldots, \mathcal{S}_n$ expresadas en lenguajes $\mathcal{L}_{\mathcal{S}_1}, \ldots, \mathcal{L}_{\mathcal{S}_n}$ sobre alfabetos $\mathcal{A}_{\mathcal{S}_1}, \ldots, \mathcal{A}_{\mathcal{S}_n}$, respectivamente. Cada alfabeto $\mathcal{A}_{\mathcal{S}_i}$ incluye un símbolo para cada elemento de la fuente $\mathcal{S}_i$, $i = 1, \ldots, n$.*

- *$\mathcal{M}$ es un conjunto de n correspondencias $\mathcal{M}_1, \ldots, \mathcal{M}_n$ entre $\mathcal{G}$ y $\mathcal{S}_1, \ldots, \mathcal{S}_n$, respectivamente. Cada correspondencia $\mathcal{M}_i$ está constituida por un conjunto de aserciones de la forma $q_\mathcal{G} \rightsquigarrow q_{\mathcal{S}_i}$, donde $q_\mathcal{G}$ y $q_{\mathcal{S}_i}$ son dos consultas de la misma aridad, respectivamente definidas sobre la ontología global $\mathcal{G}$ y $\mathcal{S}_i$, $i = 1, \ldots, n$. Las consultas $q_\mathcal{G}$ son expresadas en un lenguaje $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ sobre el alfabeto $\mathcal{A}_\mathcal{G}$ y las consultas $q_{\mathcal{S}_i}$ son expresadas en un lenguaje $\mathcal{L}_{\mathcal{M},\mathcal{S}_i}$ sobre el alfabeto $\mathcal{A}_{\mathcal{S}_i}$.*

Intuitivamente, una aserción $q_\mathcal{G} \rightsquigarrow q_{\mathcal{S}_i}$ especifica que el concepto representado por la consulta $q_\mathcal{G}$ sobre la ontología global corresponde al concepto representado por la consulta $q_{\mathcal{S}_i}$ sobre la fuente i. Las ontologías fuente describen la estructura de las fuentes, donde se hallan los datos reales, mientras que la ontología global brinda una vista integrada de las fuentes de datos subyacentes. Por otro lado, las aserciones en las correspondencias establecen la conexión entre los elementos de las ontologías globales y aquéllas en las ontologías fuente.

Las consultas al sistema $\mathcal{I}$ son realizadas en términos de la ontología globlal $\mathcal{G}$ y son expresadas en un lenguaje de consultas $\mathcal{L}_\mathcal{Q}$ sobre el alfabeto $\mathcal{A}_\mathcal{G}$. Una consulta tiene el objeto de proveer la especificación de qué pertenencias de individuos a clases se deben verificar con respecto a la ontología virtual representada por el sistema de integración de ontologías.

Efectivización del sistema de integración de ontologías basado en DL y DeLP

Ahora, efectivizaremos el sistema de integración de ontologías presentado más arriba utilizando como lenguaje de representación de ontologías a las Lógicas para la Descripción. Luego, cuando las ontologías involucradas son inconsistentes, interpretaremos a las mismas como programas de la Programación en Lógica Rebatible.

Una base de datos para una terminología dada se corresponderá con una caja asercional, que será un conjunto de hechos donde los nombres de clases, relaciones e individuos están definidos sobre un alfabeto $\mathcal{A}_\mathfrak{T}$ que coincidirá con el de la terminología asociada.

Para efectivizar el sistema de integración de ontologías $\mathcal{I} = (\mathcal{G}, \mathcal{S}, \mathcal{M})$, consideraremos un *conjunto de bases de datos fuente* para $\mathcal{I}$; esto es, un conjunto de bases de datos $\mathcal{D}_i, (i = 1, \ldots, n)$ tal que:

- Cada $\mathcal{D}_i$ es consistente. Esto es, $\mathcal{D}_i \nvdash \bot$, lo que en la práctica significa que no hay dos pares de aserciones $a : C$ y $a : \neg C$ (o $\langle a, b\rangle : P$ y $\langle a, b\rangle : \neg P$, respectivamente) en $\mathcal{D}_i$, para un concepto C, un nombre de propiedad P y nombres de constantes individuales a, b.

- Cada $\mathcal{D}_i$ es consistente con respecto a Σ. Esto es, $\Sigma \cup \mathcal{D}_i \nvdash \bot$.

Definición 4.6.2 (Sistema efectivo de integración de ontologías) *Un sistema efectivo de integración de ontologías $\mathcal{I}^*$ es una tripla $(\mathcal{G}, \mathcal{S}, \mathcal{M})$ donde:*

- $\mathcal{G}$ *es una ontología global expresada como una* δ*-ontología sobre un alfabeto* $\mathcal{A}_\mathcal{G}$*.*
- $\mathcal{S}$ *es un conjunto de* n *ontologías fuente locales* $\mathcal{S}_1, \ldots, \mathcal{S}_n$ *expresadas en lenguajes como* δ*-ontologías sobre alfabetos* $\mathcal{A}_{\mathcal{S}_1}, \ldots, \mathcal{A}_{\mathcal{S}_n}$*, respectivamente. Cada alfabeto* $\mathcal{A}_{\mathcal{S}_i}$ *incluye un símbolo para cada elemento de la fuente* $\mathcal{S}_i$*,* $i = 1, \ldots, n$*.*
- $\mathcal{M}$ *es un conjunto de* n *correspondencias* $\mathcal{M}_1, \ldots, \mathcal{M}_n$ *entre* $\mathcal{G}$ *y* $\mathcal{S}_1, \ldots, \mathcal{S}_n$*, respectivamente. Cada correspondencia* $\mathcal{M}_i$ *está constituida por un conjunto de aserciones de la forma* $q_{\mathcal{S}_i} \sqsubseteq q_\mathcal{G}$*, donde* $q_\mathcal{G}$ *y* $q_{\mathcal{S}_i}$ *son dos consultas de la misma aridad, respectivamente definidas sobre la ontología global* $\mathcal{G}$ *y* $\mathcal{S}_i$*,* $i = 1, \ldots, n$*. Las consultas* $q_\mathcal{G}$ *son expresadas el lenguaje DL sobre el alfabeto* $\mathcal{A}_\mathcal{G}$ *y las consultas* $q_{\mathcal{S}_i}$ *son expresadas en un lenguaje DL sobre el alfabeto* $\mathcal{A}_{\mathcal{S}_i}$*. Los conjuntos* $\mathcal{M}_1, \ldots, \mathcal{M}_n$ *serán llamados ontologías puente.*

Ahora, mostraremos un caso de estudio describiendo una situación donde tenemos varias *ontologías locales*, desarrolladas independientemente una de otra, posiblemente incompletas e inconsistentes, y, además, tenemos una *ontología* global como un medio para extraer información a partir de las ontologías locales, también posiblemente inconsistentes e incompletas.

Ejemplo 4.6.1 *Consideremos la* δ*-ontología global* $\mathcal{G} = (\emptyset, T_D^\mathcal{G}, \emptyset)$ *presentada en la Figura 4.13. La terminología rebatible* $T_D^\mathcal{G}$ *expresa que los genios de las computadoras usualmente no son buenos en los deportes pero que los nadadores sí lo son; si alguien es capaz de nadar un estilo de competición o uno de rescate, o es un buzo, entonces es considerado un nadador experto.*

Ontología global $\mathcal{G} = (\emptyset, T_D^\mathcal{G}, \emptyset)$**:**

$$T_D^\mathcal{G} = \left\{ \begin{array}{l} geek \sqsubseteq \neg good \\ swimmer \sqsubseteq good \\ (\exists can\_swim.rescue\_stroke \sqcap \exists can\_swim.race\_stroke) \sqcup diver \\ \qquad\qquad \sqsubseteq swimmer \end{array} \right\}$$

Figura 4.13: Ontología global $\mathcal{G}$

Note que la terminología $T_D^\mathcal{G}$ expresa un conflicto con respecto al concepto "*good*". Si descubriéramos que para un individuo de la Abox se puede probar que es un miembro de los conceptos "*swimmer*" y "*geek*", entonces la Abox sería inconsistente desde un punto de vista DL tradicional porque dicho individuo pertenecería a la vez a los conceptos "*good*" y "$\neg$*good*, indicando que el concepto "*good*" debiera ser vacío y contener un individuo a la misma vez. Mostraremos cómo esta situación puede ser manejada naturalmente en DeLP.

Ejemplo 4.6.2 (Continúa el Ejemplo 4.6.1) *En la Figura 4.14 (pág. 144) , se presentan dos ontologías fuente: $\mathcal{S}_1$, que habla sobre actividades acuáticas, y $\mathcal{S}_2$, que define términos sobre programación de computadoras. La ontología fuente local $\mathcal{S}_1 = (\emptyset, T_D^{\mathcal{S}_1}, A^{\mathcal{S}_1})$ define el significado de términos acerca de la natación y del buceo. La terminología $T_D^{\mathcal{S}_1}$ dice que los buzos con tanque y en apnea son buzos, los buzos de saturación son una clase particular de buzos con tanque, y que quien es capaz de nadar algún estilo es un nadador. También establece que los estilos de natación son crawl y side*[15]*. La caja asercional $A^{\mathcal{S}_1}$ provee la información concreta de la fuente de datos; dice que John nada los estilos crawl y side, y, además, que Paul es un buzo de saturación.*

La ontología fuente local $\mathcal{S}_2 = (T_S^{\mathcal{S}_2}, T_D^{\mathcal{S}_2}, A^{\mathcal{S}_2})$ habla de los lenguajes de programación. La terminología estricta $T_S^{\mathcal{S}_2}$ dice que entre las clases de lenguajes de programación podemos encontrar los lenguajes de programación en lógica y los orientados a objetos. La terminología rebatible $T_D^{\mathcal{S}_2}$ dice un programador es quien puede programar usando algún lenguaje de programación, y que alguien usualmente programa en un lenguaje si es capaz de leer y escribir código en dicho lenguaje a menos que haya desaprobado el curso de programación elemental. La caja asercional $A^{\mathcal{S}_2}$ establece que Prolog es un lenguaje de programación en lógica y que John puede leer y escribir código escrito en Prolog; también que Java es un lenguaje orientado a objetos; que Mary puede leer y escribir código en Java, y, Paul lee y escribe código Java pero desaprobó el curso de programación elemental.

Note como, en particular, la ontología fuente $\mathcal{S}_2$ es inconsistente pues el individuo Paul pertenece al mismo tiempo a los conceptos "*programmer*" y "$\neg$*programmer*", indicando que la Abox A_2 es incoherente. Por lo tanto, esta ontología no puede tampoco ser procesada por razonadores DL estándar.

Como dijimos previamente, nuestro objetivo es el de contestar consultas sobre pertenencia de individuos a un cierto concepto de una ontología global utilizando los datos definidos en las ontologías fuente. La relación de la ontología global con las locales se realiza a través de *ontologías puente*. Una ontología puente permite corresponder conceptos y relaciones entre dos ontologías. Así, un concepto en una ontología corresponde a una *vista* de otro u otros conceptos en otra ontología. Dichas ontologías puente implementarán la correspondencia definida más arriba. En los ejemplos presentados, consideramos dadas a las ontologías puente como *dadas*; sin embargo, existen técnicas semi-automatizadas para el descubrimiento de tales correspondencias implementadas como reglas de articulación (*e.g.*, (Mitra, 2004; Wiesman and Roos, 2004; van Diggelen, 2007)).

Ejemplo 4.6.3 (Continúa el Ejemplo 4.6.2) *Considere nuevamente la ontología global $\mathcal{G}$ y las dos ontologías fuente $\mathcal{S}_1$ y $\mathcal{S}_2$. El enlace entre las definiciones de la terminología definida en $\mathcal{G}$ con las definidas en $\mathcal{S}_1$ y $\mathcal{S}_2$ se realiza mediante las ontologías puente $\mathcal{M}_1$ y*

[15]El estilo "*side*" (o de lado) se conoce vulgarmente en Argentina como estilo "over".

Ontología fuente $\mathcal{S}_1 = (\emptyset, T_D^{\mathcal{S}_1}, A^{\mathcal{S}_1})$ sobre *Natación y buceo*:

$$T_D^{\mathcal{S}_1} = \left\{ \begin{array}{l} free\_diver \sqcup scuba\_diver \sqsubseteq diver \\ saturation\_diver \sqsubseteq scuba\_diver \\ \exists swims.stroke \sqsubseteq swimmer \end{array} \right\}$$

$$A^{\mathcal{S}_1} = \left\{ \begin{array}{ll} crawl : stroke; & side : stroke \\ \langle john, crawl \rangle : swims; & \langle john, side \rangle : swims \\ paul : saturation\_diver & \end{array} \right\}$$

Ontología fuente $\mathcal{S}_2 = (T_S^{\mathcal{S}_2}, T_D^{\mathcal{S}_2}, A^{\mathcal{S}_2})$ sobre *Lenguajes de programación*:

$$T_S^{\mathcal{S}_2} = \left\{ \; lp\_language \sqcup oop\_language \sqsubseteq language \; \right\}$$

$$T_D^{\mathcal{S}_2} = \left\{ \begin{array}{l} \exists programs.language \sqsubseteq programmer \\ \exists programs.language \sqcap failed\_programming\_101 \sqsubseteq \neg programmer \\ reads \sqcap writes \sqsubseteq programs \end{array} \right\}$$

$$A^{\mathcal{S}_2} = \left\{ \begin{array}{ll} prolog : lp\_language; & java : oop\_language \\ \langle john, prolog \rangle : writes; & \langle john, prolog \rangle : reads \\ \langle mary, java \rangle : reads; & \langle mary, java \rangle : writes \\ \langle paul, java \rangle : reads; & \langle paul, java \rangle : writes \\ paul : failed\_programming\_101 & \end{array} \right\}$$

Figura 4.14: Ontologías fuente $\mathcal{S}_1$ y $\mathcal{S}_2$

$\mathcal{M}_2$, respectivamente (ver la Figura 4.15). En dichas ontologías, los nombres de conceptos y propiedades se califican con el nombre de la ontología que los define.

La ontología puente $\mathcal{M}_1$ expresa que el concepto "swims" definido en $\mathcal{S}_1$ corresponde al concepto "can_swim" de $\mathcal{G}$; el concepto "diver" de $\mathcal{S}_1$ es "diver" en $\mathcal{G}$; el concepto (anónimo o enumerado) compuesto del individuo "side" de $\mathcal{S}_1$ es el concepto "rescue_stroke" de $\mathcal{G}$, y, el concepto compuesto del individuo "crawl" de $\mathcal{S}_1$ es el concepto "race_stroke" de $\mathcal{G}$. La ontología puente $\mathcal{M}_2$ indica que el concepto "programmer" definido en $\mathcal{S}_2$ corresponde al concepto "geek" definido en $\mathcal{G}$.

Note que, de acuerdo a las definiciones dadas en la Sección 2.5.2, los mapeos propuestos en el Ejemplo 4.6.3 corresponden a una *vista sensata*, en el sentido que, por ejemplo, el conjunto de los buzos en la ontología $\mathcal{G}$ puede ser más grande que el conjunto de los buzos en la ontología $\mathcal{S}_1$ (el mismo análisis se aplica a los otros conceptos y roles alineados).

Como se definió previamente, en el enfoque de vista global a la integración de ontologías, se realizan consultas a una ontología global la que sirve como medio para acceder a los datos que se encuentran en las ontología locales. A continuación, veremos cómo responder a la tarea de chequeo de instancia en un sistema de integración de ontologías

Ontología puente entre las ontologías $\mathcal{G}$ y $\mathcal{S}_1$:

$$\mathcal{M}_1 = \left\{ \begin{array}{l} \mathcal{S}_1 : swims \sqsubseteq \mathcal{G} : can\_swim \\ \mathcal{S}_1 : diver \sqsubseteq \mathcal{G} : diver \\ \mathcal{S}_1 : \{side\} \sqsubseteq \mathcal{G} : rescue\_stroke \\ \mathcal{S}_1 : \{crawl\} \sqsubseteq \mathcal{G} : race\_stroke \end{array} \right\}$$

Ontología puente entre las ontologías $\mathcal{G}$ y $\mathcal{S}_2$:

$$\mathcal{M}_2 = \left\{ \; \mathcal{S}_2 : programmer \sqsubseteq \mathcal{G} : geek \; \right\}$$

Figura 4.15: Ontologías puente para la integración de ontologías *Global-as-view*

cuando las ontologías involucradas pueden contener inconsistencias.

Tareas de razonamiento sobre sistemas de integración de ontologías

Para realizar tareas de inferencia en sistemas de integración de ontologías, a continuación damos la semántica de un sistema tal en términos de un programa DeLP. Luego, extenderemos las definiciones de pertenencia de individuos a conceptos para el caso de un sistema de integración de ontologías.

Definición 4.6.3 (Interpretación de un sistema efectivo de integración de ontologías) *Sea $\mathcal{I}^* = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ un sistema efectivo de integración de ontologías tal que $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ y $\mathcal{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$, donde:*

- $\mathcal{G} = (T_S^{\mathcal{G}}, T_D^{\mathcal{G}}, A^{\mathcal{G}})$;
- $\mathcal{S}_i = (T_S^{\mathcal{S}_i}, T_D^{\mathcal{S}_i}, A_i^{\mathcal{S}_i})$, *con* $i = 1, \ldots, n$, *y,*
- $\mathcal{M}_i = (T_S^{\mathcal{M}_i}, T_D^{\mathcal{M}_i})$, *con* $i = 1, \ldots, n$.

El sistema $\mathcal{I}^*$ se interpreta como el programa DeLP $\mathcal{I}^*_{DeLP} = (\Pi, \Delta)$, donde:

$$\Pi = \left(\mathfrak{T}_\Pi(T_S^{\mathcal{G}}) \right) \cup \left(\mathfrak{T}_\Pi(A^{\mathcal{G}}) \right) \cup \left(\bigcup_{i=1}^n \mathfrak{T}_\Pi(T_S^{\mathcal{S}_i}) \right) \cup \left(\bigcup_{i=1}^n \mathfrak{T}_\Pi(T_S^{\mathcal{M}_i}) \right), \; y$$

$$\Delta = \left(\mathfrak{T}_\Delta(T_D^{\mathcal{G}}) \right) \cup \left(\bigcup_{i=1}^n \mathfrak{T}_\Delta(T_D^{\mathcal{S}_i}) \right) \cup \left(\bigcup_{i=1}^n \mathfrak{T}_\Delta(T_D^{\mathcal{M}_i}) \right).$$

Ejemplo 4.6.4 (Continúa el Ejemplo 4.6.3) *La interpretación como programas DeLP de las ontologías global $\mathcal{G}$, fuentes $\mathcal{S}_1$ y $\mathcal{S}_2$, y puentes $\mathcal{M}_1$ y $\mathcal{M}_2$ se muestran en las Figuras 4.16, 4.17, y 4.18, respectivamente. La ontología global $\mathcal{G}$ es interpretada como el programa DeLP $\mathcal{P}_\mathcal{G} = (\Pi_\mathcal{G}, \Delta_\mathcal{G})$; la ontología fuente $\mathcal{S}_2$ es interpretada como el programa DeLP $\mathcal{P}_2 = (\Pi_2, \Delta_2)$; la ontología fuente $\mathcal{S}_3$ es interpretada como el programa DeLP*

$\mathcal{P}_3 = (\Pi_3, \Delta_3)$; *la ontología puente* $\mathcal{M}_1$ *es interpretada como el conjunto de reglas rebatibles* $\Delta_{\mathcal{M}_1}$; *la ontología puente* $\mathcal{M}_2$ *es interpretada como el conjunto de reglas rebatibles* $\Delta_{\mathcal{M}_2}$, *y, la ontología puente* $\mathcal{M}_3$ *es interpretada como el conjunto de reglas rebatibles* $\Delta_{\mathcal{M}_3}$.

Así la interpretación del sistema de integración de ontologías será el programa DeLP $\mathcal{P}_{\mathcal{I}^*} = (\Pi, \Delta)$ *donde:*

$$\begin{aligned} \Pi &= \Pi_{\mathcal{G}} \cup \Pi_1 \cup \Pi_2 \cup \Pi_3 \\ \Delta &= \Delta_{\mathcal{G}} \cup \Delta_1 \cup \Delta_2 \cup \Delta_3 \cup \Delta_{\mathcal{M}_1} \cup \Delta_{\mathcal{M}_2} \cup \Delta_{\mathcal{M}_3}. \end{aligned}$$

Programa DeLP $\mathcal{P}_{\mathcal{G}} = (\emptyset, \Delta_{\mathcal{G}})$ obtenido a partir de $\mathcal{G}$:

$$\Delta_{\mathcal{G}} = \left\{ \begin{array}{l} \sim good(X) \prec geek(X) \\ good(X) \prec swimmer(X) \\ swimmer(X) \prec \\ \qquad can\_swim(X,Y), rescue\_stroke(Y), \\ \qquad can\_swim(X,Z), race\_stroke(Z) \\ swimmer(X) \prec diver(X) \end{array} \right\}$$

Figura 4.16: Programa DeLP $\mathcal{P}$ obtenidos a partir de la ontología $\mathcal{G}$

Las inferencias posibles en la ontología integrada DL $\mathcal{I}^*_{DeLP}$ son modeladas por medio de un análisis dialéctico en el programa DeLP obtenido al aplicar la traducción de cada una de las ontologías DL en DeLP. Los argumentos garantizados obtenidos a partir de $\mathcal{I}^*_{DeLP}$ serán considerados las consecuencias de $\mathcal{I}^*$, provisto que la información estricta en $\mathcal{I}^*_{DeLP}$ sea consistente. Formalmente:

Definición 4.6.4 (Pertenencia potencial de individuos a conceptos) *Sea* $\mathcal{I}^* = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ *un sistema efectivo de integración de ontologías. Sea a un nombre de individuo y sea C un nombre de concepto definido en* $\mathcal{G}$. *El individuo a pertenece potencialmente al concepto C si y sólo si existe un argumento* $\mathcal{A}$ *a favor del literal* $\mathcal{G} : C(a)$ *con respecto al programa DeLP* $\mathcal{I}^*_{DeLP}$.

Definición 4.6.5 (Pertenencia justificada de individuos a conceptos) *Sea* $\mathcal{I}^* = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ *un sistema efectivo de integración de ontologías. Sea a un nombre de individuo y sea C un nombre de concepto definido en* $\mathcal{G}$. *El individuo a pertenece justificadamente al concepto C si y sólo si existe un argumento garantizado* $\mathcal{A}$ *a favor del literal* $\mathcal{G} : C(a)$ *con respecto al programa DeLP* $\mathcal{I}^*_{DeLP}$.

Definición 4.6.6 (Pertenencia estricta de individuos a conceptos) *Sea* $\mathcal{I}^* = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ *un sistema efectivo de integración de ontologías. Sea a un nombre de individuo y sea C*

Programa DeLP $\mathcal{P}_1 = (\Pi_1, \Delta_1)$ obtenido a partir de $\mathcal{S}_1$:

$$\Pi_1 = \left\{ \begin{array}{ll} stroke(crawl); & stroke(side) \\ swims(john, crawl); & swims(john, side) \\ saturation\_diver(paul) & \end{array} \right\}$$

$$\Delta_1 = \left\{ \begin{array}{l} diver(X) \prec free\_diver(X) \\ diver(X) \prec scuba\_diver(X) \\ scuba\_diver(X) \prec saturation\_diver(X) \\ swimmer(X) \prec swims(X, Y), stroke(Y) \end{array} \right\}$$

Programa DeLP $\mathcal{P}_2 = (\Pi_2, \Delta_2)$ obtenido a partir de $\mathcal{S}_2$:

$$\Pi_2 = \left\{ \begin{array}{ll} lp\_language(prolog); & oop\_language(java) \\ reads(john, prolog); & writes(john, prolog) \\ reads(mary, java); & writes(mary, java) \\ reads(paul, java); & writes(paul, java) \\ failed\_programming\_101(paul) & \\ language(X) \leftarrow lp\_language(X) & \\ \sim lp\_language(X) \leftarrow \sim language(X) & \\ language(X) \leftarrow oop\_language(X) & \\ \sim oop\_language(X) \leftarrow \sim language(X) & \end{array} \right\}$$

$$\Delta_2 = \left\{ \begin{array}{l} programmer(X) \prec programs(X, Y), language(Y) \\ \sim programmer(X) \prec \\ \qquad programs(X, Y), language(Y), failed\_programming\_101(X) \\ programs(X, Y) \prec reads(X, Y), writes(X, Y) \end{array} \right\}$$

Figura 4.17: Programas DeLP $\mathcal{P}_1$ y $\mathcal{P}_2$ obtenidos a partir de las ontologías $\mathcal{S}_1$ y $\mathcal{S}_2$

un nombre de concepto definido en $\mathcal{G}$. El individuo a pertenece estrictamente al concepto C *si y sólo si existe un argumento $\emptyset$ a favor del literal $\mathcal{G} : C(a)$ con respecto al programa DeLP $\mathcal{I}^*_{DeLP}$.*

Definición 4.6.7 (Pertenencia indeterminada de un individuo a una clase) *Sea $\mathcal{I}^* = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ un sistema efectivo de integración de ontologías. Sea a un nombre de individuo y sea C un nombre de concepto definido en $\mathcal{G}$. La pertenencia de un individuo a a una clase C está indeterminada si no existe ningún argumento a favor del literal $\mathcal{G} : C(a)$ con respecto al programa DeLP $\mathcal{I}^*_{DeLP}$.*

Nótese como etiquetamos al nombre de concepto C con el nombre de la ontología $\mathcal{G}$ a la que pertenece siguiendo la convención de espacios de nombres de XML.[16]

[16]Esta convención es presentada en (Haase and Motik, 2005).

Reglas puente entre las ontologías $\mathcal{G}$ y $\mathcal{S}_1$:

$$\Delta_{\mathcal{M}_1} = \left\{ \begin{array}{l} \mathcal{G} : can\_swim(X,Y) \prec \mathcal{S}_1 : swims(X,Y) \\ \mathcal{G} : diver(X) \prec \mathcal{S}_1 : diver(X) \\ \mathcal{G} : rescue\_stroke(X) \prec \mathcal{S}_1 : stroke(X), X = side \\ \mathcal{G} : race\_stroke(X) \prec \mathcal{S}_1 : stroke(X), X = crawl \end{array} \right\}$$

Reglas puente entre las ontologías $\mathcal{G}$ y $\mathcal{S}_2$:

$$\Delta_{\mathcal{M}_2} = \left\{ \; \mathcal{G} : geek(X) \prec \mathcal{S}_2 : programmer(X) \; \right\}$$

Figura 4.18: Reglas puente expresadas como reglas rebatibles

A continuación mostraremos algunos de los argumentos que pueden ser construidos a partir del sistema integrado de ontologías. En el resto de la presentación asumiremos la especificidad generalizada como criterio de comparación de argumentos.

Ejemplo 4.6.5 (Continúa el Ejemplo 4.6.4) *Considere nuevamente el programa DeLP $\mathcal{P}_{\mathcal{I}^*}$ presentado en el Ejemplo 4.6.4, bajo este programa, estamos interesados en determinar la pertenencia garantizada de los individuos John, Mary y Paul a los conceptos "good" y/o "¬good". De acuerdo a la Definición 4.6.5, es necesario determinar si existen argumentos garantizados para los literales good(john), good(mary) y good(paul), respectivamente. Debido a que en DeLP no es posible que existan argumentos garantizados y complementarios simultáneamente, las respuestas a las consultas nunca serán ambiguas. Veremos que como John es un genio de la computación y un nadador, no será posible determinar si es o no bueno en los deportes. A pesar de este resultado desalentador, veremos también que como Mary es un programador Java, ella no será considerada buena en los deportes. En el caso de Paul, como es un buzo de saturación y, a pesar de que programa en Java pero desaprobó el curso elemental de programación, no será considerado un programador y, así, será considerado bueno en los deportes.*

Primero, consideraremos el análisis dialéctico para la consulta "good(john)". Existen razones para creer que John pertenece potencialmente al concepto "good". Formalmente,

existe un argumento $\langle \mathcal{A}_1, good(john)\rangle$ *donde:*

$$\mathcal{A}_1 = \left\{ \begin{array}{l} (good(john) \prec swimmer(john)), \\ (swimmer(john) \prec \\ \quad can\_swim(john, side), rescue\_stroke(side), \\ \quad can\_swim(john, crawl), race\_stroke(crawl)), \\ (\mathcal{G} : race\_stroke(crawl) \prec \\ \quad \mathcal{S}_1 : stroke(crawl), crawl = crawl), \\ (\mathcal{G} : can\_swim(john, crawl) \prec \\ \quad \mathcal{S}_1 : swims(john, crawl)), \\ (\mathcal{G} : rescue\_stroke(side) \prec \\ \quad \mathcal{S}_1 : stroke(side), side = side) \\ (\mathcal{G} : can\_swim(john, side) \prec \\ \quad \mathcal{S}_1 : swims(john, side)) \end{array} \right\}.$$

Sin embargo, como John pertenece potencialmente al concepto "$\neg good$*" porque es un genio de la computación. Formalmente, hay un argumento* $\langle \mathcal{A}_2, \sim good(john)\rangle$ *que derrota al argumento* $\mathcal{A}_1$*, donde:*

$$\mathcal{A}_2 = \left\{ \begin{array}{l} (\sim good(john) \prec geek(john)), \\ (\mathcal{G} : geek(john) \prec \mathcal{S}_2 : programmer(john)), \\ (programmer(john) \prec \\ \quad programs(john, prolog), language(prolog)), \\ (programs(john, prolog) \prec \\ \quad reads(john, prolog), writes(john, prolog)) \end{array} \right\}.$$

Así, en el árbol dialéctico para la consulta "$good(john)$*", el argumento derrotado* $\mathcal{A}_1$ *aparece etiquetado com un nodo-D mientras que el argumento victorioso* $\mathcal{A}_2$ *aparece marcado como un nodo-U (ver Fig. 4.19.(a)). Por otro lado, cuando consideramos la pertenencia de John al concepto "*$\neg good$*", descubrimos que el argumento* $\mathcal{A}_2$ *que sustenta esta conclusión es derrotado por el argumento* $\mathcal{A}_1$ *(ver Fig. 4.19.(b)). Por lo tanto, la respuesta a la consulta "*$good(john)$*" es* INDECISO.

Segundo, consideremos el análisis dialéctico para determinar si Mary pertenece al concepto "good". Mary pertenece justificadamente al concepto "$\neg good$*" ya que la respuesta a la consulta "*$good(mary)$*" es* NO *porque existe un argumento garantizado* $\langle \mathcal{B}_1, \sim good(mary)\rangle$ *(ver la Figura 4.20), donde:*

$$\mathcal{B}_1 = \left\{ \begin{array}{l} (\sim good(mary) \prec geek(mary)), \\ (\mathcal{G} : geek(mary) \prec \mathcal{S}_2 : programmer(mary)), \\ (programmer(mary) \prec \\ \quad programs(mary, java), language(java)), \\ (language(java) \prec oop\_language(java)), \\ (programs(mary, java) \prec \\ \quad reads(mary, java), writes(mary, java)) \end{array} \right\}.$$

Tercero, veremos el porqué Paul pertenece justificadamente al concepto "good". Consideremos el árbol dialéctico para el literal "good(paul)". Existe un argumento $\langle \mathcal{C}_1, good(paul) \rangle$, *basado en la información rebatible que expresa que Paul es un nadador experto (pues es un buzo de saturación), con:*

$$\mathcal{C}_1 = \left\{ \begin{array}{l} (good(paul) \prec swimmer(paul)), \\ (swimmer(paul) \prec \mathcal{G} : diver(paul)), \\ (\mathcal{G} : diver(paul) \prec \mathcal{S}_1 : diver(paul)), \\ (\mathcal{S}_1 : diver(paul) \prec scuba\_diver(paul)), \\ (scuba\_diver(paul) \prec saturation\_diver(paul)) \end{array} \right\}.$$

Pero el argumento $\mathcal{C}_1$ *es atacado por un argumento* $\langle \mathcal{C}_2, \sim good(paul) \rangle$, *donde:*

$$\mathcal{C}_2 = \left\{ \begin{array}{l} (\sim good(paul) \prec geek(paul)), \\ (\mathcal{G} : geek(paul) \prec \mathcal{S}_2 : programmer(paul)), \\ (programmer(paul) \prec \\ \quad programs(paul, java), language(java)), \\ (programs(paul, java) \prec \\ \quad reads(paul, java), writes(paul, java)) \end{array} \right\}.$$

Este argumento dice que Paul no debe ser bueno en los deportes pues es un genio de la computación (ya que programa en Java y, por lo tanto, es un programador). Sin embargo, Paul también pertenece potencialmente al concepto "$\mathcal{S}_2 : \neg programmer$*" (porque desaprobó el curso de programación elemental), ya que se puede hallar un argumento* $\langle \mathcal{C}_3, \mathcal{S}_2 : \sim programmer(paul) \rangle$, *donde:*

$$\mathcal{C}_3 = \left\{ \begin{array}{l} (\sim programmer(paul) \prec \\ \quad programs(paul, java), language(java), \\ \quad failed\_programming\_101(paul)), \\ (programs(paul, java) \prec \\ \quad reads(paul, java), writes(paul, java)) \end{array} \right\}.$$

Así, el árbol dialéctico para la consulta "good(paul)" tiene tres nodos (ver la Figura 4.21.(a)). Con respecto a la consulta "$\sim good(paul)$*" para determinar si Paul pertenece justificadamente al concepto "*$\neg good$*", el argumento* $\mathcal{C}_2$ *es derrotado por el argumento* $\mathcal{C}_1$ *(ver la Figura 4.21.(b)). Por lo tanto, la respuesta a la consulta "good(paul)" es* SI, *y concluimos que Paul pertenece justificadamente al concepto "good".*

4.6.2. Integración de ontologías de abajo hacia arriba

Como se explicó en la Sección 2.5.3, en el acercamiento *local-as-view* (LAV) a la integración de ontologías, se asume que se posee una (o varias) ontología local $\mathcal{S}$ y una ontología global $\mathcal{G}$. En la ontología local, ciertos elementos de la misma (conceptos y/o

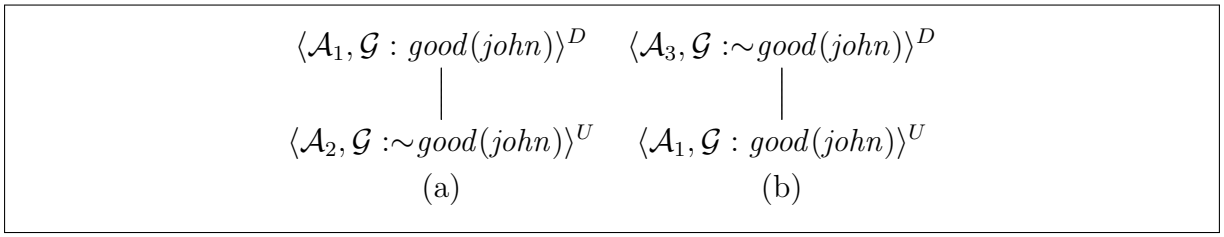


Figura 4.19: Análisis dialéctico para determinar si John pertenece al concepto *good*

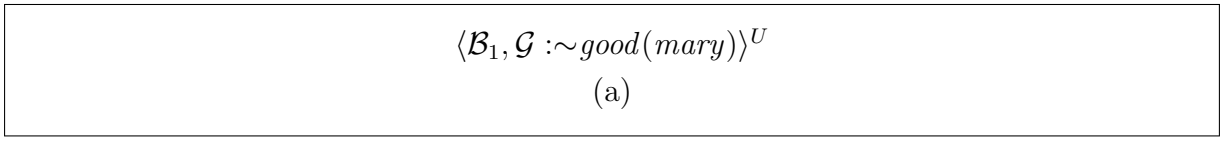


Figura 4.20: Análisis dialéctico para determinar si Mary pertenece al concepto *good*

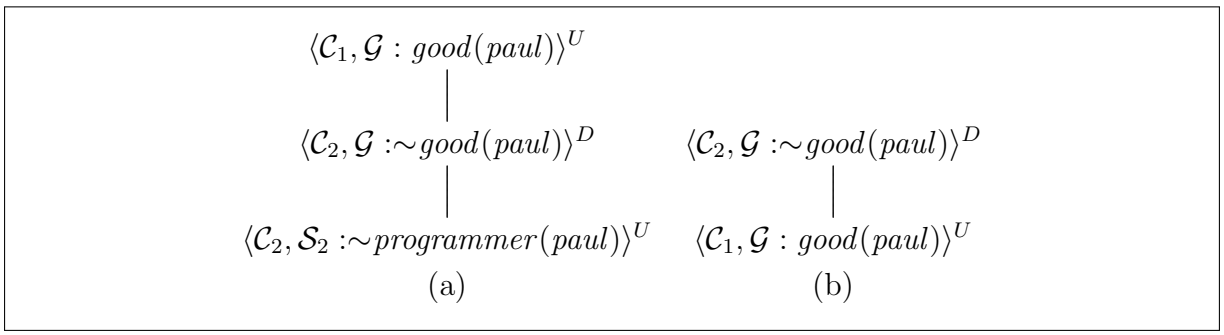


Figura 4.21: Análisis dialéctico para determinar si Paul pertenece al concepto *good*

roles) son definidos en términos de vistas o consultas definidas sobre una ontología global. Así, el significado de la asociación entre un elemento C de $\mathcal{S}$ y de una consulta $q_{\mathcal{G}}$ sobre $\mathcal{G}$ es que tal consulta representa la mejor manera de caracterizar las instancias de C usando los conceptos en $\mathcal{G}$. En esta sección, veremos cómo adaptar la integración de ontologías con el modelo LAV cuando las mismas son posiblemente inconsistentes utilizando la argumentación rebatible.

En los acercamientos GAV y LAV a la integración de datos, las consultas con respecto a la ontología blanco (en inglés, *target*) son reformuladas con respecto a las fuentes, (Haase and Motik, 2005) al referirse al trabajo de (Lenzerini, 2002) explican que, en los sistemas GAV, el problema se reduce simplemente a desplegar (en inglés, *unfolding*) las vistas, ya que la reformulación está explícita en las correspondencias. En el caso LAV, el problema requiere más pasos de razonamiento complejos ya que en el caso de los mapeos sensatos no es claro cómo reexpresar los conceptos de una ontología fuente en función de una ontología global. Por eso, en este trabajo de Tesis, restringiremos el caso de la integración LAV a vistas de una ontología local en función de una ontología global que satisfaga solamente la *completitud* (véase la Sección 2.5.3).

Formalmente, un sistema de integración de ontologías bajo el modelo LAV será definido de la misma manera que en el caso GAV (véase la Definición 4.6.2). La diferencia fundamental estribará en la forma en la cual se definen las correspondencias utilizadas para alinear los conceptos y roles de la/s ontología/s local/es con la global.

Definición 4.6.8 (Sistema efectivo de integración de ontologías LAV) *Un sistema efectivo de integración de ontologías $\mathcal{I}^*$ es una tripla $(\mathcal{G}, \mathcal{S}, \mathcal{M})$ donde:*

- *$\mathcal{G}$ es una ontología global expresada como una δ-ontología sobre un alfabeto $\mathcal{A}_{\mathcal{G}}$.*
- *$\mathcal{S}$ es un conjunto de n ontologías locales $\mathcal{S}_1, \ldots, \mathcal{S}_n$ expresadas en lenguajes como δ-ontologías sobre alfabetos $\mathcal{A}_{\mathcal{S}_1}, \ldots, \mathcal{A}_{\mathcal{S}_n}$, respectivamente. Cada alfabeto $\mathcal{A}_{\mathcal{S}_i}$ incluye un símbolo para cada elemento de la fuente $\mathcal{S}_i$, $i = 1, \ldots, n$.*
- *$\mathcal{M}$ es un conjunto de n correspondencias $\mathcal{M}_1, \ldots, \mathcal{M}_n$ entre $\mathcal{G}$ y $\mathcal{S}_1, \ldots, \mathcal{S}_n$, respectivamente. Cada correspondencia $\mathcal{M}_i$ está constituida por un conjunto de aserciones de la forma $q_{\mathcal{S}_i} \sqsubseteq q_{\mathcal{G}}$, donde:*
  - *$q_{\mathcal{G}}$ y $q_{\mathcal{S}_i}$ son dos consultas de la misma aridad, respectivamente definidas sobre la ontología global $\mathcal{G}$ y $\mathcal{S}_i$, $i = 1, \ldots, n$.* [17]
  - *Las consultas $q_{\mathcal{G}}$ son expresadas en el lenguaje DL $\mathcal{L}_h$ sobre el alfabeto $\mathcal{A}_{\mathcal{G}}$ y las consultas $q_{\mathcal{S}_i}$ son expresadas en un lenguaje DL $\mathcal{L}_b$ sobre el alfabeto $\mathcal{A}_{\mathcal{S}_i}$.*

[17] Cuando decimos que las consultas tienen la misma aridad, queremos expresar que mapean o bien conceptos (*i.e.*, predicados de aridad uno) o propiedades (*i.e.*, predicados de aridad dos).

- *Las consultas q_{S_i} corresponden a un concepto de la ontología local C o su complemento $\neg C$ o a un rol P.*
- *Los conjuntos $\mathcal{M}_1, \ldots, \mathcal{M}_n$ serán llamados ontologías puente.*

Nótese que, de acuerdo a las definiciones dadas en la Sección 2.5.3, el tipo de los mapeos propuestos en la definición dada arriba corresponden a una *vista sensata*, en el sentido que, por ejemplo, el conjunto de los individuos perteneciendo a un concepto en una ontología local $\mathcal{S}_i, i = 1, \ldots, n$ puede ser más pequeño (*i.e.*, con menor aridad), pero nunca más grande, que el conjunto de los individuos determinado por el concepto relacionado por una vista $q_{\mathcal{G}} \sqsubseteq q_{\mathcal{S}_i}$ en la ontología $\mathcal{G}$.

Entonces, en el contexto de un sistema de integración de ontologías LAV, y asumiendo una interpretación del sistema de integración de ontologías como la brindada en la Definición 4.6.3, las nociones de pertenencias estricta, potencial y justificada de un individuo a un concepto en una ontología local se definen como se da a continuación:

Definición 4.6.9 (Pertenencia potencial de individuos a conceptos) *Sea $\mathcal{I}^* = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ un sistema efectivo de integración de ontologías. Sea a un nombre de individuo y sea C un nombre de concepto definido en $\mathcal{S}_i, i = 1, \ldots, n$. El individuo a pertenece potencialmente al concepto C si y sólo si existe un argumento $\mathcal{A}$ a favor del literal $\mathcal{G} : C(a)$ con respecto al programa DeLP $\mathcal{I}^*_{DeLP}$.*

Definición 4.6.10 (Pertenencia justificada de individuos a conceptos) *Sea $\mathcal{I}^* = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ un sistema efectivo de integración de ontologías. Sea a un nombre de individuo y sea C un nombre de concepto definido en $\mathcal{S}_i, i = 1, \ldots, n$.. El individuo a pertenece justificadamente al concepto C si y sólo si existe un argumento garantizado $\mathcal{A}$ a favor del literal $\mathcal{G} : C(a)$ con respecto al programa DeLP $\mathcal{I}^*_{DeLP}$.*

Definición 4.6.11 (Pertenencia estricta de individuos a conceptos) *Sea $\mathcal{I}^* = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ un sistema efectivo de integración de ontologías. Sea a un nombre de individuo y sea C un nombre de concepto definido en $\mathcal{S}_i, i = 1, \ldots, n$. El individuo a pertenece estrictamente al concepto C si y sólo si existe un argumento $\emptyset$ a favor del literal $\mathcal{G} : C(a)$ con respecto al programa DeLP $\mathcal{I}^*_{DeLP}$.*

Definición 4.6.12 (Pertenencia indeterminada de un individuo a una clase) *Sea $\mathcal{I}^* = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ un sistema efectivo de integración de ontologías. Sea a un nombre de individuo y sea C un nombre de concepto definido en $\mathcal{S}_i, i = 1, \ldots, n$. La pertenencia de un individuo a a una clase C está indeterminada si no existe ningún argumento a favor del literal $\mathcal{G} : C(a)$ con respecto al programa DeLP $\mathcal{I}^*_{DeLP}$.*

Como explicamos más arriba, nótese como también en este caso etiquetamos al nombre de concepto C con el nombre de la ontología $\mathcal{S}_i$ a la que pertenece siguiendo la convención de espacios de nombres de XML. Cuando no haya duda, dejaremos de lado esta notación.

Ejemplo 4.6.6 *Consideremos el dominio del problema de la evaluación de tesis de posgrado y la formalización del mismo, que discutimos a continuación. En la Figura 4.22, presentamos una ontología global $\mathcal{G}_{4.6.6}$ donde se define el siguiente conocimiento: un profesor con título de posgrado puede ser revisor de tesis; alguien que no es profesor ni tiene título de posgrado, no puede ser revisor de tesis; sin embargo, alguien que es profesor, a pesar de no tener título de posgrado, si posee una carrera sobresaliente como investigador, se acepta como revisor de tesis.*

Además, presentamos dos ontologías locales $\mathcal{L}_1$ y $\mathcal{L}_2$ en la Figura 4.23. La ontología $\mathcal{L}_1$ expresa que John es un profesor que posee un doctorado, Paul es un profesor que no tiene ni un doctorado ni una maestría, Mary solamente tiene una maestría, y Steve no es profesor pero tiene una maestría. La correspondencia $\mathcal{M}_{\mathcal{L}_1,\mathcal{G}}$ expresa que los términos maestría y doctorado de la ontología local $\mathcal{L}_1$ están contenidos en el término posgraduado de la ontología global, y que si alguien no tiene una maestría ni un doctorado entonces no se ha posgraduado. La ontología $\mathcal{L}_2$ expresa que a es un artículo, b un libro, c un capítulo y que Paul ha publicado a a, b y c. La correspondencia $\mathcal{M}_{\mathcal{L}_2,\mathcal{G}}$ expresa que la vista correspondiente a los individuos que han publicado un artículo, un capítulo y un libro corresponden a los investigadores sobresalientes.

Al considerar la semántica de las ontologías anteriores en términos de reglas DeLP, se obtienen los programas que presentamos en las Figuras 4.24 y 4.25. Además, al considerar la pertenencia justificada de los individuos John, Paul, Mary y Steve al concepto jurado de tesis (en inglés, "reviewer") con respecto al sistema de integración de ontologías $(\mathcal{G}, \{\mathcal{L}_1, \mathcal{L}_2\}, \{\mathcal{M}_{\mathcal{L}_1,\mathcal{G}}, \mathcal{M}_{\mathcal{L}_2,\mathcal{G}}\})$ se obtienen los análisis dialécticos que se presentan a continuación:

El individuo John pertenece en forma justificada al concepto "reviewer" pues el argumento $\langle \mathcal{A}, reviewer(john) \rangle$ no tiene derrotadores y, por lo tanto, está garantizado (veáse la Figura 4.26.(a)), con:

$$\mathcal{A} = \left\{ \begin{array}{l} (reviewer(john) \prec postgrad(john), prof(john)), \\ (postgrad(john) \prec phd(john)) \end{array} \right\}.$$

En el caso de Paul, concluiremos que Paul es un posible revisor ya que el mismo pertenece en forma justificada al concepto "reviewer"; presentamos la explicación a continuación y su resumen se puede ver en la Figura 4.26.(b)-(c). Paul pertenece potencialmente al concepto "¬reviewer" pues existe un argumento $\langle \mathcal{B}_1, \sim reviewer(paul) \rangle$, con:

$$\mathcal{B}_1 = \left\{ \begin{array}{ll} (\sim reviewer(paul) \prec & \sim postgrad(paul)), \\ (\sim postgrad(paul) \prec & \sim msc(paul), \sim phd(paul)) \end{array} \right\}.$$

Sin embargo, vemos que tenemos un argumento $\langle \mathcal{B}_2, reviewer(paul) \rangle$ que derrota a $\mathcal{B}_1$, con:

$$\mathcal{B}_2 = \left\{ \begin{array}{l} (reviewer(paul) \prec prof(paul), \sim postgrad(paul), outstanding(paul)), \\ (outstanding(paul) \prec published(paul, a), article(a), \\ \qquad published(paul, b), book(b), published(paul, c), chapter(c)), \\ (\sim postgrad(paul) \prec \sim msc(paul), \sim phd(paul)) \end{array} \right\}.$$

Como $\mathcal{B}_2$ no está derrotado, concluimos que el literal $reviewer(paul)$ está garantizado.

Por otro lado, Steve no será un jurado ya que pertenece en forma justificada al concepto "$\neg reviewer$" (véase la Figura 4.26.(d)). En este caso, existe un único argumento (no derrotado) $\langle \mathcal{D}, \sim reviewer(steve) \rangle$.

En cambio, para Mary no es posible expresar ninguna afirmación con respecto a su pertenencia al concepto "reviewer" ya que no hay argumentos para el literal $reviewer(mary)$ ni para $\sim reviewer(mary)$.

Ontología global $\mathcal{G}_{\mathbf{4.6.6}} = (\emptyset, T_D^{\mathcal{G}}, \emptyset)$:

$$T_D^{\mathcal{G}} = \left\{ \begin{array}{l} prof \sqcap postgrad \sqsubseteq reviewer \\ \neg postgrad \sqcup \neg prof \sqsubseteq \neg reviewer \\ prof \sqcap \neg postgrad \sqcap outstanding \sqsubseteq reviewer \end{array} \right\}$$

Figura 4.22: Ontología global $\mathcal{G}_{4.6.6}$ para integración LAV

4.7. Discusión: Partición automatizada de Tboxes en Sboxes y Dboxes

Una de las ventajas de los lenguajes de representación de ontologías basados en las Lógicas para la Descripción, como OWL, es que pueden ser utilizados para computar relaciones de subsunción entre clases y para identificar clases insatisfacibles (inconsistentes). Sin embargo, los razonadores DL estándar basados en el algoritmo de *tableaux* (como Racer (Haarslev and Möller, 2001), Fact (Horrocks, 1998), Pellet (Parsia and Sirin, 2004)), solamente brindan una lista de clases insatisfacibles. El proceso de "depurar" la ontología (es decir, determinar qué clases son insatisfacibles) es dejado al usuario. Cuando el ingeniero de conocimiento se enfrenta con varias clases insatisfacibles en una ontología moderadamente grande, puede encontrar dificultades para hallar el error subyacente (Wang et al., 2005).

En cambio, en el acercamiento propuesto en esta Tesis, los axiomas problemáticos (es decir, aquellos que involucran definiciones de clases inconsistentes) son separados en un conjunto diferenciado llamado Dbox mientras que los axiomas no problemáticos (es decir, aquellos que no involucran excepciones a la jerarquía de clases) son conservados en un

Ontología $\mathcal{L}_1 = (\emptyset, \emptyset, A^{\mathcal{L}_1})$:

$$A^{\mathcal{L}_1} = \left\{ \begin{array}{lll} john : prof; & john : phd & \\ paul : prof; & paul : \neg msc; & paul : \neg phd \\ mary : msc; & & \\ steve : \neg prof; & steve : msc & \end{array} \right\}$$

Correspondencia $\mathcal{M}_{\mathcal{L}_1,\mathcal{G}}$ entre $\mathcal{L}_1$ y $\mathcal{G}_{\mathbf{4.6.6}}$:

$$\mathcal{M}_{\mathcal{L}_1,\mathcal{G}} = \left\{ \begin{array}{l} \mathcal{L}_1 : msc \sqcup \mathcal{L}_1 : phd \sqsubseteq \mathcal{G} : postgrad \\ \mathcal{L}_1 : \neg msc \sqcap \mathcal{L}_1 : \neg phd \sqsubseteq \mathcal{G} : \neg postgrad \end{array} \right\}$$

Ontología $\mathcal{L}_2 = (\emptyset, \emptyset, A^{\mathcal{L}_2})$:

$$A^{\mathcal{L}_2} = \left\{ \begin{array}{l} a : article \\ b : book \\ c : chapter \\ \langle paul, a \rangle : published \\ \langle paul, b \rangle : published \\ \langle paul, c \rangle : published \end{array} \right\}$$

Correspondencia $\mathcal{M}_{\mathcal{L}_2,\mathcal{G}}$ entre $\mathcal{L}_2$ y $\mathcal{G}_{\mathbf{4.6.6}}$:

$$\mathcal{M}_{\mathcal{L}_2,\mathcal{G}} = \left\{ \begin{array}{l} \mathcal{L}_2 : (\exists published.article \sqcap \exists published.book \sqcap \\ \qquad \exists published.chapter) \sqsubseteq \mathcal{G} : outstanding \end{array} \right\}$$

Figura 4.23: Ontologías locales $\mathcal{L}_1$ y $\mathcal{L}_2$ para integración LAV

conjunto llamado Sbox. En las secciones previas de nuestra presentación, tales conjuntos se han considerados como dados; es decir, el ingeniero de conocimiento decide por si mismo cómo integrar la Sbox y la Dbox. Sin embargo, cuando se involucran ontologías importadas en la discusión, no queda claro cómo realizar tal separación. Por ello, en esta sección discutimos acercamientos para realizar tal separación de una Tbox en una Sbox y una Dbox.

4.7.1. Dos acercamientos ingenuos

En esta sección presentamos dos acercamientos ingenuos a la separación de Tboxes en Sboxes y Dboxes: el primero consiste en considerar a todas los axiomas de la Tbox como rebatibles (es decir, todos forman parte de la Dbox); el segundo consiste en agrupar todas las reglas de Programación en Lógica Extendida que se obtienen a partir de una Tbox que tengan como cabeza a literales complementarios.

Programa DeLP obtenido a partir de $\mathcal{G}_{4.6.6}$:

$$\Delta_{\mathcal{G}} = \left\{ \begin{array}{l} reviewer(X) \prec postgrad(X), prof(X). \\ \sim reviewer(X) \prec \sim postgrad(X). \\ \sim reviewer(X) \prec \sim prof(X). \\ reviewer(X) \prec prof(X), \sim postgrad(X), outstanding(X). \end{array} \right\}$$

Figura 4.24: Ontología global $\mathcal{G}_{4.6.6}$ expresada como un programa DeLP

Definición 4.7.1 (δ-ontología asociada) *Dada una ontología DL $\Sigma = (T, A)$, llamaremos δ-ontología asociada a Σ, notado como $\mathfrak{Asoc}(\Sigma)$, a la δ-ontología que se obtiene a partir de Σ aplicando alguna estrategia de conversión.*

Definición 4.7.2 (Programa DeLP asociado) *Dada una ontología DL $\Sigma = (T, A)$, llamaremos programa DeLP asociado a Σ, notado como $\mathfrak{Prog}(\Sigma)$ al programa DeLP que se obtiene al traducir Σ al formalismo de la DeLP.*

La estrategia más simple para obtener la ontología asociada a una ontología expresada en Lógicas para la Descripción consiste en que todos los axiomas de la Tbox sean considerados como parte de la Dbox y que las aserciones de la Abox permanezcan como tales. Formalmente:

Estrategia 4.7.1 *Dada una ontología DL $\Sigma = (T, A)$, la δ-ontología asociada a Σ se forma como $\mathfrak{Asoc}(\Sigma) = (\emptyset, T, A)$. Además, el programa asociado a Σ se obtiene como $\mathfrak{Prog}(\Sigma) = (\mathfrak{T}_{\Pi}(A), \mathfrak{T}_{\Delta}(T))$.*

Ejemplo 4.7.1 *Considere la ontología DL tradicional $\Sigma_{4.7.1} = (T, A)$ donde: $T = \{(C \sqsubseteq D), (E \sqsubseteq \neg D)\}$ y $A = \{(a : C), (a : E)\}$. En este caso la δ-ontología asociada es $\mathfrak{Asoc}(\Sigma_{4.7.1}) = (\emptyset, T, A)$ y el programa asociado es $\mathfrak{Prog}(\Sigma_{4.7.1}) = (\{(D(X) \prec C(X)), (\sim D(X) \prec E(X))\}, \{C(a), E(a)\})$.*

Ejemplo 4.7.2 *Como ejemplo de la Estrategia 4.7.1 considere cualquiera de las δ-ontologías presentadas en los Ejemplos 4.4.2, 4.4.3, 4.4.5, o 4.4.7. En tales casos, las terminologías originales, si las hubiera, han sido expresadas como terminologías rebatibles (o Dboxes) ya que las terminologías estrictas (o Sboxes) son conjuntos vacíos.*

Propiedad 4.7.1 *Dada una ontología DL $\Sigma = (T, A)$, si la Abox A es coherente, entonces el conjunto de hechos $\mathfrak{T}_{\Pi}(A)$ es consistente.*

Demostración: *Supongamos que $\mathfrak{T}_{\Pi}(A)$ es inconsistente. Luego, existen un par de literales unarios o binarios complementarios.*

Programa DeLP $\mathcal{P}_1 = (\Pi^{\mathcal{L}_1}, \emptyset)$ obtenido de $\mathcal{L}_1$:

$$\Pi^{\mathcal{L}_1} = \left\{ \begin{array}{lll} prof(john). & phd(john). & \\ prof(paul). & \sim msc(paul). & \sim phd(paul). \\ msc(mary). & & \\ \sim prof(steve). & msc(steve). & \end{array} \right\}$$

Correspondencia $\mathcal{M}_{\mathcal{L}_1,\mathcal{G}}$ expresada en DeLP:

$$\Delta_{\mathcal{L}_1,\mathcal{G}} = \left\{ \begin{array}{l} \mathcal{G} : postgrad(X) \prec \mathcal{L}_1 : msc(X). \\ \mathcal{G} : postgrad(X) \prec \mathcal{L}_1 : phd(X). \\ \sim\mathcal{G} : postgrad(X) \prec \ \sim\mathcal{L}_1 : msc(X), \sim\mathcal{L}_1 : phd(X). \end{array} \right\}$$

Programa DeLP $\mathcal{P}_2 = (\Pi^{\mathcal{L}_2}, \emptyset)$ obtenido de $\mathcal{L}_2$:

$$\Pi^{\mathcal{L}_2} = \left\{ \begin{array}{lll} article(a). & book(b). & chapter(c). \\ published(paul, a). & published(paul, b). & published(paul, c). \end{array} \right\}$$

Correspondencia $\mathcal{M}_{\mathcal{L}_2,\mathcal{G}}$ entre $\mathcal{L}_2$ y $\mathcal{G}$ 4.6.6:

$$\Delta_{\mathcal{L}_2,\mathcal{G}} = \left\{ \begin{array}{l} \mathcal{G} : outstanding(X) \prec \\ \qquad \mathcal{L}_2 : published(X, Y), \mathcal{L}_2 : article(Y), \\ \qquad \mathcal{L}_2 : published(X, Z), \mathcal{L}_2 : book(Z), \\ \qquad \mathcal{L}_2 : published(X, W), \mathcal{L}_2 : chapter(W). \end{array} \right\}$$

Figura 4.25: Ontologías locales $\mathcal{L}_1$ y $\mathcal{L}_2$ expresadas en DeLP

Caso unario: *Supongamos que $\mathfrak{T}_\Pi(A)$ contiene dos literales complementarios $C(a)$ y $\sim C(a)$ (donde C es un nombre de concepto y a de individuo). Contradicción: La Abox A es incoherente pues contiene dos aserciones $a : C$ y $a : \neg C$.*

Caso binario: *Supongamos que $\mathfrak{T}_\Pi(A)$ contiene dos literales complementarios $r(a, b)$ y $\sim r(a, b)$ (donde R es un nombre de rol y a, b de individuos). Esta situación no puede ocurrir porque nuestro lenguaje DL no permite negación de roles (si $\sim r(a, b) \in \mathfrak{T}_\Pi(A)$, entonces $\langle a, b\rangle : \neg r \in A$).*

□

Otra estrategia para obtener en forma automática un programa lógico rebatible a partir de una ontología expresada en Lógicas para la Descripción consiste en obtener un conjunto de reglas expresadas en el lenguaje de la Programación en Lógica Extendida[18] y luego separar tales reglas en dos conjuntos de acuerdo al siguiente criterio: todas las

[18]Por *Programación en Lógica Extendida* entendemos a reglas Prolog extendidas con negación clásica $\neg$ y negación *default* not de átomos (Baral, 2003).

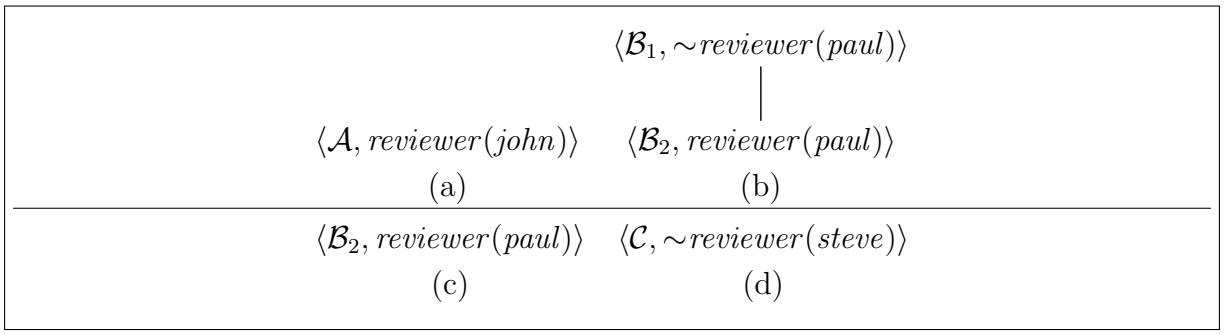


Figura 4.26: Árboles de dialéctica para la integración LAV

reglas para las cuales existen literales complementarios son reglas rebatibles y el resto son reglas estrictas. Formalmente:

Estrategia 4.7.2 *Dada una ontología DL $\Sigma = (T, A)$, sea $\mathcal{P}$ un programa lógico extendido donde el conjunto de sus hechos se define como $\mathfrak{T}_\Pi(A)$ y el conjunto de sus reglas como $\mathfrak{T}^*_\Pi(T)$. Sea $\mathfrak{Part} : 2^{\mathcal{L}_{ELP}} \to (\mathcal{L}_\Pi, \mathcal{L}_\Delta)$ tal que $\mathfrak{Part}(R) = (R_\Pi, R_\Delta)$, donde*

$$\begin{array}{lll} R_\Delta & =_{def} & \{\ r \in R : \exists r' \in R, Head(r) = \overline{Head(r')}\ \} \\ R_\Pi & =_{def} & R - R_\Delta \end{array}$$

La situación descripta por la Estrategia 4.7.2 se puede apreciar en la parte superior de la Figura 4.27.

Ejemplo 4.7.3 *Consideremos la Tbox DL $T_{4.7.3}$ tal que:*

$$T_{4.7.3} = \left\{ \begin{array}{l} (A \sqcap B) \sqcup C \sqsubseteq D \sqcap E \\ \neg A \sqcup F \sqsubseteq \neg E \end{array} \right\}.$$

A partir de esta Tbox se genera el siguiente conjunto de reglas R de la programación en lógica extendida:

$$R = \left\{ \begin{array}{l} d(X) \leftarrow a(X), b(X). \\ e(X) \leftarrow a(X), b(X). \\ d(X) \leftarrow c(X). \\ e(X) \leftarrow c(X). \\ \neg e(X) \leftarrow \neg a(X). \\ \neg e(X) \leftarrow f(X). \end{array} \right\}.$$

Así, aplicando la Estrategia 4.7.2, obtenemos los siguientes conjuntos de reglas estrictas R_Π y rebatibles R_Δ:

$$\begin{array}{lll} R_\Pi & = & \left\{ \begin{array}{l} d(X) \leftarrow a(X), b(X). \\ d(X) \leftarrow c(X). \end{array} \right\} \\ R_\Delta & = & \left\{ \begin{array}{l} e(X) \prec a(X), b(X). \\ e(X) \prec c(X). \\ \sim e(X) \prec \sim a(X). \\ \sim e(X) \prec f(X). \end{array} \right\} \end{array}$$

4.7.2. Acercamiento basado en convertir axiomas problemáticos en rebatibles

Por otro lado, si observamos nuevamente la Figura 4.27, esta vez prestando atención al borde izquierdo de la misma, notamos un segundo acercamiento para la partición automatizada de Tboxes en Sboxes y Dboxes. En este caso, es interesante desarrollar una función $\mathfrak{Sep}$ que sea capaz de realizar la separación de axiomas de inclusión en dos conjuntos: uno de axiomas de inclusión estrictos y otro de axiomas de inclusión rebatibles. Luego, estos dos conjuntos serán transformados en reglas estrictas y rebatibles como se lo ha planteado en secciones previas (mostrado en la sección inferior de la Figura 4.27).

Un número importante de ontologías OWL están apareciendo en la Web Semántica y los lenguajes de ontologías se han vuelto más expresivos. Sin embargo, esto hace que la resolución de inconsistencias en la Web Semántica sea una tarea desafiante para los modeladores de ontologías. Los servicios de razonamiento de las Lógicas para la Descripción estándar pueden verificar si una ontología es satisfacible; sin embargo, no proveen soporte para resolver la insatisfacibilidad. Otros acercamientos se basan en identificar los axiomas problemáticos (detectando las subontologías minimalmente insatisfacibles) o debilitando las ontologías insatisfacibles (hallando todas las subontologías maximalmente satisfacibles).

Ejemplo 4.7.4 (Tomado de (Lam et al., 2006)) *Asuma que una ontología* Σ *contiene los siguientes axiomas:*

$$\begin{array}{lcl} r_1 & : & A \equiv C \sqcap D \sqcap G \\ r_2 & : & C \equiv E \sqcap F \\ r_3 & : & E \equiv \neg D \sqcap (\exists R.D) \\ r_4 & : & H \equiv \forall R.C \end{array}$$

Utilizando el razonamiento DL para Tboxes estándar, se puede demostrar que el concepto es insatisfacible. Los acercamientos estándar o bien identifican la ontología mínimamente insatisfacible $\Sigma_{min} = \{r_1, r_2, r_3\}$ *o calculan las subontologías maximalmente satisfacibles* $\Sigma_{max_1} = \{r_2, r_3, r_4\}$, $\Sigma_{max_2} = \{r_1, r_3, r_4\}$ *y* $\Sigma_{max_3} = \{r_1, r_2, r_4\}$. *En resumen, uno de los axiomas* r_1, r_2 *o* r_3 *debe ser removido de* Σ. *Sin embargo, es posible mostrar que no es necesario remover alguno de los axiomas de arriba "completos". De hecho, se puede simplemente remover alguno de las siguientes partes de los axiomas: (i)* $A \sqsubseteq C$, *(ii)* $A \sqsubseteq D$, *(iii)* $C \sqsubseteq E$, *o (iv)* $E \sqsubseteq \neg D$, *y de esta manera* Σ *se vuelve satisfacible.*

Lam et al. (2006) presentan un algoritmo que extiende el algoritmo de tableaux de Meyer et al. (2006). El algoritmo no solamente ubica los axiomas problemáticos, sino que también rastrea qué partes de los axiomas son responsables de la insatisfacibilidad de un concepto destino. Usando este algoritmo, primero se calculan las deducciones perdidas de los conceptos con nombre debido a la remoción de axiomas. Cuando una parte de

un axioma o de un axioma completo es removido, frecuentemente ocurre que muchos de las deducciones pretendidas son perdidas. Para minimizar el impacto sobre la ontología, se analizan las deducciones perdidas (*e.g.*, subsunción de conceptos) de conceptos con nombre debido a la remoción de axiomas. Entonces, utilizando el algoritmo de Lam et al. (2006) como una "caja negra", proponemos la siguiente estrategia para separar una Tbox en una Sbox y una Dbox.

Definición 4.7.3 (Conjunto de conceptos asociados a un axioma / terminología) *Sean $r_1, \ldots, r_n$ un conjunto de axiomas tales que forman una terminología $T = \{r_1, \ldots, r_n\}$. Definimos $\mathfrak{Concepts}(r_i)$ como el conjunto de los conceptos con nombre del axioma r_i. Extendemos la definición al conjunto de los conceptos con nombre de una terminología T tal que:*

$$\mathfrak{Concepts}(T) = \bigcup_{i=1,\ldots,n} \mathfrak{Concepts}(r_i)$$

Definición 4.7.4 (Algoritmo de Lam *et al.*) *Sea T una terminología y C un nombre de concepto en T. Definimos como algoritmo de Lam a la función $\mathfrak{Lam}_C(T)$ que toma una terminología T y un nombre de concepto C y retorna el conjunto de los conceptos con nombre en T relevantes a la insatisfacibilidad del concepto C.*

Estrategia 4.7.3 *Sea $\Sigma = (T, A)$ una ontología DL. Sea $C \in \mathfrak{Concepts}(\Sigma)$. Sea $\mathfrak{Lam}_C(T) = \{c_1, \ldots, c_n\} \subseteq \mathfrak{Concepts}(T)$ el conjunto de conceptos relevantes a la insatisfacibilidad del concepto C. La función de separación de una Tbox en una Sbox y una Dbox, notada como $\mathfrak{Sep}(T) = (T_S, T_D)$ se define como:*

$$\begin{array}{rcl} T_D & = & \left\{\ r \in T : (\mathfrak{Concepts}(r) \cap \mathfrak{Lam}_C(T)) \neq \emptyset\ \right\} \\ T_S & = & T - T_D \end{array}$$

Ejemplo 4.7.5 *Considere la siguiente terminología $T_{4.7.5}$ (tomada de (Lam et al., 2006)):*

$$T_{4.7.5} = \left\{ \begin{array}{l} r_1 : \mathit{Bird}^* \sqsubseteq \mathit{Fly}^* \sqcap \mathit{Animal} \\ r_2 : \mathit{Eagle} \sqsubseteq \mathit{Bird} \\ r_3 : \mathit{Penguin}^* \sqsubseteq \mathit{Bird}^* \sqcap (\neg \mathit{Fly})^* \end{array} \right\}.$$

Siguiendo la notación de (Lam et al., 2006), en esta ontología los conceptos etiquetados con una estrella son relevantes a la insatisfacibilidad del concepto "Penguin"; por lo tanto,

$$\mathfrak{Lam}_{\mathit{Penguin}}(T_{4.7.5}) = \left\{\ \mathit{Bird}, \mathit{Fly}, \mathit{Penguin}, (\neg \mathit{Fly})\ \right\}.$$

Así, en lugar de eliminar todos los axiomas que contienen referencias a conceptos etiquetados con una estrella, lo cual daría lugar a la pérdida de ciertas inferencias, proponemos convertir dichos axiomas en rebatibles. Así tal Tbox $T_{4.7.5}$ se partiría en una Sbox $T_S = \{r_2\}$ y una Dbox $T_D = \{r_1, r_3\}$.

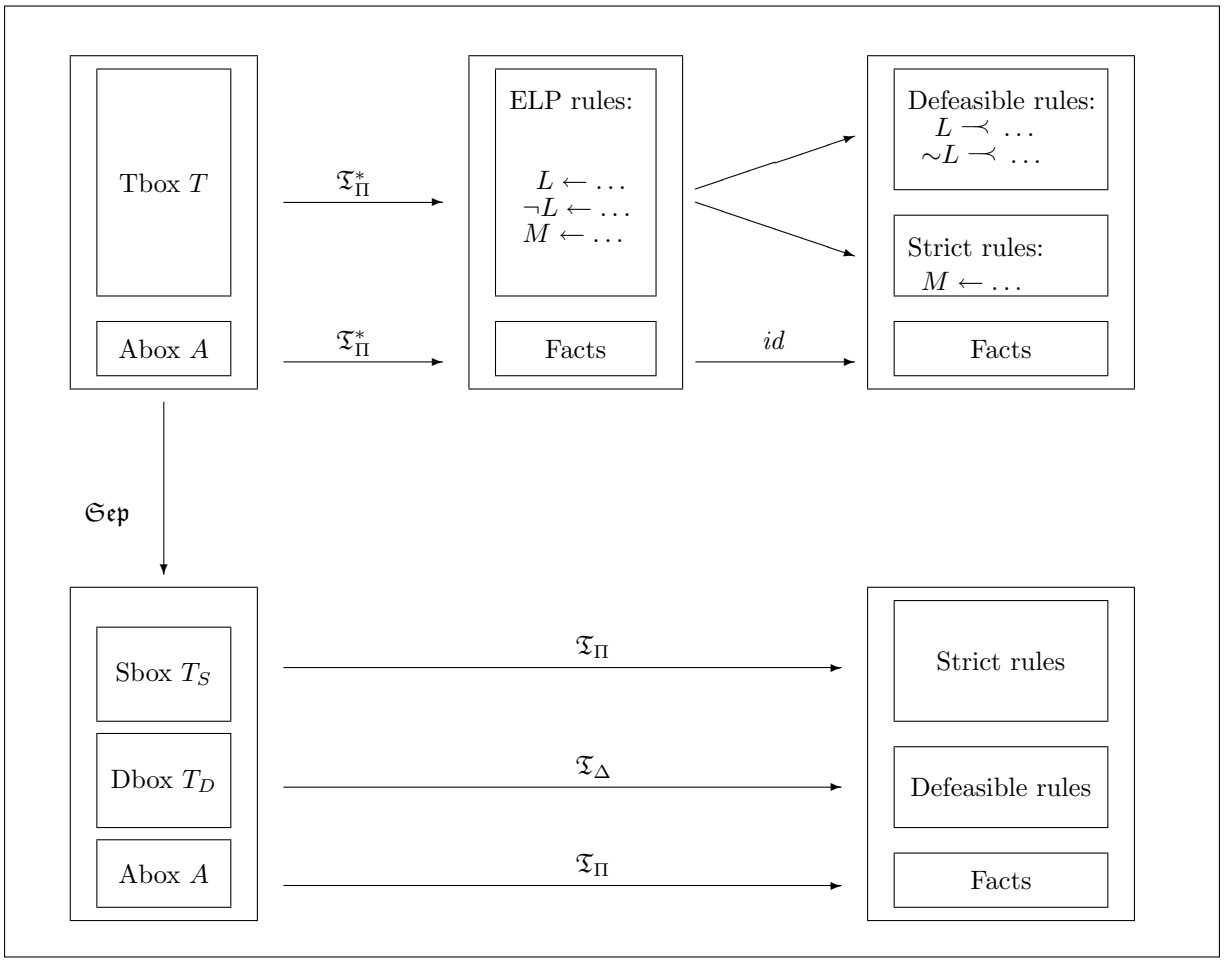


Figura 4.27: Obtención de reglas estrictas y rebatibles a partir de una ontología DL

4.8. Trabajo relacionado

4.8.1. DLP: Grosof, Motik & Volz

Grosof *et al.* (Grosof et al., 2003; Volz, 2004) muestran como interoperar, semántica e inferencialmente, entre los acercamientos a la Web Semántica basados en reglas (*e.g.*, *Programas Lógicos RuleML*) y ontologías (Lógicas para la Descripción OWL/DAML+OIL) a través del análisis de su intersección expresiva. Para realizar tal tarea, definen un nuevo lenguaje intermedio de representación de conocimiento contenido en dicha intersección: los *Programas Lógicos Descriptivos* (en inglés, *Description Logic Programs* o DLP), y la cercanamente relacionada *Lógica de Horn para la Descripción* (en inglés, *Description Horn Logic* o DHL) el cual es un fragmento expresivo de la Lógica de Primer Orden. Muestran cómo realizar la llamada *Fusión-DLP*, la cual consiste de la traducción bidireccional de premisas e inferencias (incluyendo las típicas consultas de las DL) a partir del fragmento

DLP de las DL a LP, y viceversa a partir del fragmento DLP de la Programación en Lógica a las DL.

Volz muestra como traducir ciertos axiomas DL a un programa lógico manteniendo el significado de los mismos, presentando también una técnica novedosa para preprocesar axiomas DL de tal manera que la traducción a la programación en lógica sea posible. El preprocesamiento remueve las redundancia en la sintaxis y remueve también la negación de los axiomas Tbox (tanto como es posible). Las negaciones restantes son interpretadas como negación por falla finita.

DLP-convert

Para facilitar el uso de DLP, Motik *et al.*(Motik et al., 2003, 2005) brindan la herramienta de conversión `dlpconvert` que permite convertir fragmentos DLP codificados en OWL en sintaxis de *Edimburgh Prolog*. `dlpconvert`[19] está basado en los algoritmos para reducir Lógicas para la Descripción a Datalog implementado en KAON2. El sistema lee una ontología OWL, la reduce a Datalog disyuntivo, si es posible, y finalmente la serializa como un programa lógico, el cual puede ser usado para una lectura y entendimiento más fácil para personas con una formación apropiada en lógica e intérpretes Prolog.

A continuación presentamos un ejemplo basado en (Motik et al., 2005).

Ejemplo 4.8.1 (Basado en (Motik et al., 2005)) *Consideremos el clásico ejemplo de lógica:*

> *Todos los humanos son mortales.*
>
> *Socrates es humano.*
>
> *Por lo tanto, Socrates es mortal.*

La siguiente ontología OWL expresa que todos los humanos son mortales y que Socrates es humano:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Ontology [
  <!ENTITY ex
      "http://logic.uni-karlsruhe.de/dlpconvert/example1#">]>
<owlx:Ontology owlx:name="&ex;"
          xmlns:owlx="http://www.w3.org/2003/05/owl-xml#">
<owlx:Class owlx:name="#human" owlx:complete="false">
<owlx:Class owlx:name="#mortal"/>
</owlx:Class>
<owlx:Individual owlx:name="#Socrates">
           <owlx:type owlx:name="#human" />
</owlx:Individual>
</owlx:Ontology>
```

[19] `dlpconvert` viene como una herramienta de línea de comando, implementada en Java 5.

Utilizando la herramienta `dlpconvert`*, esta ontología es traducida como se espera dando lugar al siguiente código Prolog:*

```
mortal(X) :- human(X).
human(socrates).
```

También como es de esperarse, para realizar la consulta para saber si Socrates es mortal, escribiremos:

```
?- mortal(socrates).
yes
```

Discusión: Parte de nuestro acercamiento está basado en el trabajo de (Grosof et al., 2003; Volz, 2004). Las funciones de traducción de DLs a DeLP son una extensión del trabajo presentado por Grosof y Volz. Sin embargo, Grosof y Volz se limitan a reglas Prolog estándar (aumentadas con negación por falla finita); es decir, no consideran la negación clásica como parte de su lenguaje de representación de conocimiento. Por lo tanto, no son capaces de lidiar con bases de conocimiento DL inconsistentes como sí lo hace nuestro acercamiento.

4.8.2. Acercamiento con Answer Set Programming: Eiter

Un requerimiento clave para la arquitectura en capas de la Web Semántica es la integración de las capas de Reglas y Ontologías. En particular, esto es crucial para permitir la construcción de reglas sobre las ontologías, *i.e.* lograr que sistemas basados en reglas usen vocabulario especificado en bases de conocimiento ontológicas. En este sentido, Eiter *et al.* (Eiter et al., 2004) proponen una combinación de la Programación en Lógica bajo la Semántica de Conjuntos de Repuestas (en inglés, *Answer Set Semantics*) con las Lógicas para la Descripción $\mathcal{SHIF}(D)$ y $\mathcal{SHOIN}(D)$, las cuales subyacen a los lenguajes OWL-Lite y OWL-DL, respectivamente. Esta combinación permite construir reglas sobre las ontologías, y en una extensión limitada, construir ontologías sobre las reglas. Introducen los *programas lógicos descriptivos* o *dl-programas* (en inglés, *description logic programs* o *dl-programs*), los cuales consisten de una base de conocimiento DL L y un conjunto finito de *dl-reglas* (en inglés, *description logic rules* o *dl-rules*) P. Tales reglas son similares a las reglas usuales en los programas lógicos con negación por falla, pero también pueden contener *consultas* a L, posiblemente negadas con negación *default*, en sus cuerpos. En tal consulta, se pregunta si un cierto axioma DL (o su negación) se deduce o no a partir de L. Una *dl-consulta* (en inglés, *dl-query*) $Q(t)$ es uno de los siguientes: (a) un axioma de inclusión de conceptos F o su negación $\neg F$; (b) de las formas $C(t)$ o $\neg C(t)$, donde C es un concepto y t es un término; (c) de las formas $R(t_1, t_2)$ o $\neg R(t_1, t_2)$, donde R es un rol y t_1, t_2 son términos. Un *dl-átomo* (en inglés, *dl-atom*) tiene la forma:

$DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](t)$, con $m \geq 0$, donde cada S_i es o bien un concepto o un rol, $op_i \in \{\uplus, \cup, \cap\}$, p_i es un símbolo de predicado unario (resp. binario), y $Q(t)$ es una *dl-consulta*. Llaman a $p_1, \ldots, p_m$ sus *símbolos de predicados de entrada*. Intuitivamente, $op_i = \uplus$ (resp., $op_i = \cup$) aumenta S_i (resp., $\neg S_i$) por la extensión de p_i, mientras que $op_i = \cap$ restringe S_i a p_i. Una *dl-regla* r tiene la forma $a \leftarrow b_1, \ldots, b_k, \mathsf{not}\, b_{k+1}, \ldots, \mathsf{not}\, b_m$, $m \geq k \geq 0$, donde $a, b_1, \ldots, b_m$ son literales clásicos; además, cualquier literal $b_1, \ldots, b_m$ perteneciendo al cuerpo de r puede ser de ser un *dl-átomo*. Un *dl-programa* $KB = (L, P)$ consiste de una base de conocimiento DL y un conjunto finito de *dl-reglas* P. El siguiente ejemplo ilustra las ideas principales de este acercamiento:

Ejemplo 4.8.2 (Tomado de (Eiter et al., 2004)) *Supongamos que deseamos asignar revisores a artículos, basados en cierta información acerca de los artículos y las personas disponibles, utilizando una ontología L_S (presentada parcialmente en la Figura 4.28), la cual contiene conocimiento acerca de publicaciones científicas. Asumimos que no poseemos conocimiento acerca de la estructura completa y el contenido de L_S salvo los siguientes aspectos. L_S clasifica artículos en áreas de investigación en base a palabras clave. Las áreas de investigación están contenidas en un concepto $Area$. Los roles $keyword$ e $inArea$ asocian a cada artículo sus palabras claves relevantes y las áreas en las cuales están clasificados. Además, un rol $expert$ relaciona las personas con su área de especialidad, y un concepto $Referee$ contiene todos los revisores. Finalmente, un rol $hasMember$ asocia un grupo de palabras claves similares con todos sus miembros.* [20]

Consideremos ahora el siguiente dl-programa $KB_S = (L_S, P_S)$, donde P_S contiene en particular las siguientes dl-reglas:

(1) $paper(p_1); kw(p_1, Semantic\_Web);$
(2) $paper(p_2); kw(p_1, Bioinformatics); kw(p_2, Answer\_Set\_Programming);$
(3) $kw(P, K_2) \leftarrow kw(P, K_1), DL[hasMember](S, K_1), DL[hasMember](S, K_2);$
(4) $paperArea(P, A) \leftarrow DL[keyword \uplus kw; inArea](P, A);$
(5) $cand(X, P) \leftarrow paperArea(P, A), DL[Referee](X), DL[expert](X, A);$
(6) $assign(X, P) \leftarrow cand(X, P), \mathsf{not}\, \neg assign(X, P);$
(7) $\neg assign(Y, P) \leftarrow cand(X, P), assign(X, P), X \neq Y;$
(8) $a(P) \leftarrow assign(X, P);$
(9) $error(P) \leftarrow paper(P), \mathsf{not}\, a(P)$

Intuitivamente, las reglas (1) y (2) especifican la información de palabras claves de dos artículos, p_1 y p_2, los cuales deberían ser asignados a revisores. La regla (3) aumenta, por elección del diseñador, la información de claves con palabras similares. La regla (4) consulta el conjunto L_S aumentado para recuperar las áreas en las que cada artículo es clasificado. La regla (5) selecciona candidatos a revisar artículos basados en esta información a partir de expertos entre los revisores de acuerdo a L_S. Las reglas (6) y (7) eligen

[20] Las clases D_{string} y $D_{\mathbb{N}}$ representan los dominios concretos de las cadenas de texto y los números naturales respectivamente (véase la Sección 2.3.4).

Ontología L_S:

$\geq 1\ title \sqsubseteq Publication$;
$\top \sqsubseteq \forall title.D_{string}$;
$\geq 1\ year \sqsubseteq Publication$;
$\top \sqsubseteq \forall year.D_{\mathbb{N}}$;
$\geq 1\ firstname \sqsubseteq Person$;
$\top \sqsubseteq \forall firstname.D_{string}$;
$\geq 1\ lastname \sqsubseteq Person$;
$\geq 1\ keyword \sqsubseteq Paper$;
$\geq 1\ cites \sqsubseteq Paper$;
$\geq 1\ contains \sqsubseteq Area$;
$\geq 1\ hasAuthor \sqsubseteq Paper$;
$\geq 1\ expert \sqsubseteq Person$;
$\geq 1\ inArea \sqsubseteq Paper$;
$\geq 1\ hasMember \sqsubseteq TopicCluster$;
$isContainedIn \equiv contains^{-}$;
$isAuthorOf \equiv hasAuthor^{-}$;
$isMemberOf \equiv hasMember^{-}$;
$Paper \sqsubseteq Publication$;
$Referee \sqsubseteq Person$;
$\exists inArea.\{A\} \equiv \exists keyword.(\exists isContainedIn.\{A\})$;
$\exists expert.\{A\} \equiv \exists isAuthorOf.(\exists inArea.\{A\})$;
$\exists inArea.\{B\} \equiv \exists keyword.(\exists isContainedIn.\{B\})$;
$\exists expert.\{B\} \equiv \exists isAuthorOf.(\exists inArea.\{B\})$;
$\exists inArea.\{C\} \equiv \exists keyword.(\exists isContainedIn.\{C\})$;
$\exists expert.\{C\} \equiv \exists isAuthorOf.(\exists inArea.\{C\})$;
$\exists inArea.\{D\} \equiv \exists keyword.(\exists isContainedIn.\{D\})$;
$\exists expert.\{D\} \equiv \exists isAuthorOf.(\exists inArea.\{D\})$;

$\top \sqsubseteq \forall lastname.D_{string}$;
$\top \sqsubseteq \forall keyword.Kw$;
$\top \sqsubseteq \forall cites.Paper$;
$\top \sqsubseteq \forall contains.Kw$;
$\top \sqsubseteq \forall hasAuthor.Person$;
$\top \sqsubseteq \forall expert.Area$;
$\top \sqsubseteq \forall inArea.Area$;
$\top \sqsubseteq \forall hasMember.Kw$;

$Kw(Belief\_Revision)$;
$Kw(Frame\_Systems)$;
$Kw(Intelligent\_Agents)$;
$Kw(Bioinformatics)$;
$Area(A)$; $Area(B)$;
$Area(C)$; $Area(D)$;
$contains(A, Belief\_Revision)$;
$contains(A, Default\_Reasoning)$;
$contains(B, Frame\_Systems)$;
$contains(B, Ontologies)$;
$contains(C, Semantic\_Web)$;

Figura 4.28: Ontología L_S (tomada de (Eiter et al., 2004))

uno de los revisores candidatos para un artículo. Finalmente, las reglas (8) y (9) verifican si cada artículo es asignado; en caso contrario, se dispara una bandera de error.

La dl-regla siguiente muestra cómo se pueden codificar restricciones numéricas calificadas, las cuales no se encuentran disponibles en $\mathcal{SHOIN}(D)$; define un experto como un autor de al menos tres artículos en la misma área:

$$\begin{aligned} expert(X, A) \quad \leftarrow \quad & DL[isAuthorOf](X, P_1), \\ & DL[isAuthorOf](X, P_2), \\ & DL[isAuthorOf](X, P_3), \\ & DL[inArea](P_1, A), \\ & DL[inArea](P_2, A), \\ & DL[inArea](P_3, A), \\ & P_1 \neq P_2, P_2 \neq P_3, P_3 \neq P_1. \end{aligned}$$

En resumen, Eiter *et al.* proponen una combinación de ontologías y reglas en la Web Semántica. Dicha combinación es lograda a través de la integración de una ontología $\mathcal{SHOIQ}(D)$ L_S con un programa AnsProlog P_S. La ontología L_S no posee ninguna característica especial. En cambio, el programa P_S presenta como una variante interesante

el agregado de operaciones especiales de acceso a la ontología L_S. Dichas operaciones van ubicadas en el cuerpo de la regla y permiten verificar afirmación de chequeo de instancia (en inglés, *instance checking*) sobre L_S así como la recuperación de datos de L_S para ser utilizados en la evaluación de reglas en P_S.

Entre las ventajas del acercamiento de Eiter *et al.* se puede mencionar que, en la forma en que las consultas hacia las ontologías DL subyacentes son manejadas, el acercamiento provisto permite encapsular el acceso a dichas ontologías ya que las inferencias en los programas lógicos y en las ontologías se hallan separadas. Esta característica aumenta la flexibilidad del acercamiento y lo hace apropiado para respetar cuestiones de privacidad tanto en los programas lógicos como en las ontologías consultadas. Por otro lado, al resolver las consultas propuestas con respecto a un conjunto de reglas también se retiene el no determinismo de la programación en lógica.

Al utilizar la maquinaria de AnsProlog y por lo tanto utilizar el operador de negación *default*, el acercamiento de Eiter *et al.* permite razonar acerca de no-monotonicidad de reglas lógicas. Sin embargo, dicho acercamiento no permite razonar con ontologías DL subyacentes. Para solucionar este problema, el acercamiento de Eiter puede ser adaptado al marco de DeLP en varias formas:

1. disponer de un programa DeLP $\mathcal{P}$ que contenga en algunos de los literales de los cuerpos de las reglas consultas a una ontología DL L_P estándar subyacente;

2. considerar tener un programa DeLP $\mathcal{P}$ que posea en algunos de los literales de los cuerpos de las reglas consultas a un δ-ontología Σ;

3. combinar una δ-ontología Σ con una ontología DL tradicional L_P, en este caso Σ en en los miembros izquierdos de los axiomas de inclusión puede contener consultas a L_P;

4. definir δ-ontologías con consultas a otras δ-ontologías (este caso es equivalente a la integración de ontologías presentado en este capítulo), o,

5. definir ontologías DL con consultas a programas DeLP (este caso es equivalente a construir ontologías apoyadas en reglas).

En el ejemplo presentado a continuación y para respetar el espíritu de trabajo de Eiter *et al.* exploraremos la opción (1).

Ejemplo 4.8.3 *Considere el siguiente programa DeLP* $\mathcal{P}_S = (\Pi, \Delta)$ *que establece reglas para la asignación de revisores a artículos enviados para su consideración a una cierta conferencia, donde* Π *y* Δ *son tales que:*

$$\Pi = \left\{ \begin{array}{l} paper(p_1); \\ kw(p_1, Semantic\_Web); \\ paper(p_2); \\ kw(p_1, Bioinformatics); \\ kw(p_2, Answer\_Set\_Programming) \end{array} \right\}$$

$$\Delta = \left\{ \begin{array}{l} kw(P, K_2) \prec kw(P, K_1), DL[hasMember](S, K_1), DL[hasMember](S, K_2) \\ paperArea(P, A) \prec DL[keyword \uplus kw; inArea](P, A) \\ cand(X, P) \prec paperArea(P, A), DL[Referee](X), DL[expert](X, A) \\ a(P) \prec assign(X, P) \\ assign(X, P) \prec cand(X, P) \\ \sim assign(X, P) \prec cand(X, P), DL[isAuthorOf](Y, P), \\ \qquad DL[inst](X, I_1), DL[inst](Y, I_2), I_1 = I_2 \end{array} \right\}$$

El conjunto Π *tiene el mismo sentido que las definiciones dadas por Eiter; las cuatro primeras reglas del conjunto* Δ *también tienen el mismo sentido que en las definiciones de Eiter. El cambio aparece en las dos últimas reglas del conjunto* Δ*: la penúltima regla dice que un artículo* P *es asignado a un revisor* X *si el revisor* X *es candidato para* P*, la última regla expresa que un artículo* P *no es asignado a* X *si el revisor* X *es candidato a revisar pero alguno de los autores* Y *trabaja en la misma institución que el revisor.*

Supongamos que Sergio y Martín son autores de artículos del área C *(véase la Figura 4.28) correspondiente a la Web Semántica y ambos trabajan en el laboratorio LIDIA. Supongamos que Sergio es un experto en el tema de la Web Semántica. Puede ser el caso en que Sergio sea elegido como evaluador para un artículo enviado por Martín. Sin embargo, esta situación es poco ética ya que ambos trabajan en el mismo laboratorio. Veamos cómo esta situación puede ser resuelta naturalmente en DeLP utilizando el programa rebatible* $\mathcal{P}_S$*. A partir de* $\mathcal{P}_S$ *surge el siguiente análisis dialéctico: Es posible construir un argumento* $\langle \mathcal{A}, cand(\text{sergio}, p_1) \rangle$ *donde:*

$$\mathcal{A} = \left\{ \begin{array}{l} assign(\text{sergio}, p_1) \prec cand(\text{sergio}, p_1) \\ cand(\text{sergio}, p_1) \prec \\ \qquad paperArea(p_1, c), DL[Referee](\text{sergio}), DL[expert](\text{sergio}, c) \\ paperArea(p_1, c) \prec DL[keyword \uplus kw; inArea](p_1, c) \end{array} \right\}$$

Pero este argumento tiene un derrotador $\langle \mathcal{B}, \sim cand(\text{sergio}, p_1) \rangle$ *con:*

$$\mathcal{B} = \mathcal{A} \cup \left\{ \begin{array}{l} \sim assign(\text{sergio}, p_1) \prec cand(\text{sergio}, p_1), DL[isAuthorOf](\text{martin}, p_1), \\ \qquad DL[inst](p_1, lidia\_lab), DL[inst](\text{martin}, lidia\_lab), \\ \qquad lidia\_lab = lidia\_lab \end{array} \right\}$$

En consecuencia Sergio no será asignado como revisor de un artículo de Martín (ni viceversa).

4.8.3. Mitra: *Ontology-Composition Algebra*

Mitra (Mitra et al., 2000; Mitra, 2004; Mitra and Wiederhold, 2001, 2002) presenta un *álgebra de composición de ontologías* (*Ontology-Composition Algebra*) que consiste de un conjunto de operadores básicos que pueden ser usados para manipular ontologías. Los operadores algebráicos pueden ser usados para especificar cómo componer nuevas ontologías derivadas a partir de ontologías fuente ya existentes. Una especificación declarativa permite la fácil reproducción de la tarea composicional cuando las ontologías fuente cambian y dichos cambios necesitan ser propagados a las ontologías derivadas. Si no existe un medio para actualizar rápida y fácilmente las ontologías derivadas cuando las ontologías fuente cambian, las ontologías derivadas suportan información desactualizada y muchas veces inconsistente de sus clientes. Antes de que múltiples ontologías puedan ser compuestas, la heterogeneidad semántica entre estas ontologías debe ser resuelta y un conjunto de reglas de articulación deben ser establecidas. Estas reglas de articulación especifican la correlación entre los conceptos relacionados entre las ontologías fuente.

Mitra ha desacoplado la maquinaria algebráica que es usada para manipular ontologías de la componente que deriva la correspondencia semántica entre ontologías para crear dos componentes distinguidas: (1) las *funciones de generación de reglas de articulación* generan reglas de articulación entre pares de ontologías, y (2) los *operadores algebraicos* usan las reglas de articulación para componer las ontologías fuente. Las funciones de generación de reglas son implementadas como subrutinas semiautomáticas que utilizan algoritmos heurísticos para articular ontologías. La evidencia empírica en (Mitra, 2004), muestra que las funciones de generación semi-automática de articulaciones pueden ser implementadas y conforman una componente útil de herramientas de composición de información. El Álgebra de Composición de Ontologías presentada por Mitra tiene operaciones unarias y binarias que permiten que un compositor de ontologías seleccione porciones interesantes de una ontologías y las componga. Mitra caracteriza las propiedades de los operadores algebráicos e identifica las condiciones necesarias y suficientes para la optimización de las tareas de composición. Muchas de estas propiedades dependen de las propiedades de la función de generación de articulaciones empleada para resolver la heterogeneidad semántica entre las ontologías que están siendo compuestas.

El formato de ontologías presentado por Mitra se llama *Onion*. Onion es un subconjunto de RDF y está fundado en un trabajo previo de Gyssens. Onion también utiliza reglas en la forma de cláusulas de Horn. En Onion, las únicas relaciones con semántica predefinida son: *subclass*, *property*, *instance* y *value*. Se pueden representar otras relaciones pero su semántica tiene que ser definida por el usuario creador de la ontología a través de reglas. Onion contiene un subsistema llamado SKAT que realiza la articulación de ontologías. Para tratar las ontologías como objetos de primera clase es necesario reificarlas.

En el acercamiento de Mitra, las reglas de articulación entre ontologías son generadas en forma semi-automática mientras que en nuestro acercamiento asumimos las reglas de

articulación como dadas. Ciertamente, el acercamiento de Mitra puede ser considerado complementario al propuesto en esta Tesis.

4.8.4. Heymans y Vermeir

En (Heymans and Vermeir, 2002), Heymans y Vermier extienden la Lógica para la Descripción $\mathcal{SHOQ}(D)$ con un orden de preferencia sobre los axiomas. Con este orden parcial estricto ciertos axiomas pueden ser desautorizados en el caso en que sean derrotados por otros axiomas más preferidos. También imponen un semántica basada en modelos preferidos; de esta manera, introducen efectivamente nomonotonicidad en $\mathcal{SHOQ}(D)$. Heymans y Vermier argumentan que ya una Lógica para la Descripción puede ser vista como un lenguaje de ontologías, o la traducción propia de uno, obtienen lo que ellos llaman un *lenguaje de ontologías rebatibles*.

A continuación, presentamos brevemente los fundamentos del acercamiento de Heymans y Vermeir. Una ontología $\langle T, \prec \rangle$ está definida por una Tbox T está compuesta de axiomas de inclusión $r_1 : A_1 \sqsubseteq B_1, \ldots, r_n : A_n \sqsubseteq B_n$, y un orden parcial estricto (y bien fundado) $\prec$ entre los axiomas. Dado un axioma $A \sqsubseteq B$, una *interpretación* $\mathcal{I}$ definida sobre un dominio de interpretación $\Delta^{\mathcal{I}}$ y un individuo x, el estado del axioma $A \sqsubseteq B$ puede estar en: (1) *aplicable* si $x \in A^{\mathcal{I}}$; (2) *aplicado* si (1) y $x \in B^{\mathcal{I}}$; (3) *clásicamente satisfecho* si vale (2) cada vez que se cumple (1), y, (4) *derrotado* si vale (2) y existe $(C \sqsubseteq D) \prec (A \sqsubseteq B)$ tal que $C \sqsubseteq D$ está aplicado. En el caso (4) se dice que "$C \sqsubseteq D$" *derrota* a "$A \sqsubseteq B$".

La noción de *modelo* se halla modificada acordemente. Una interpretación $\mathcal{I}$ *satisface* a un axioma $A \sqsubseteq B$ si cada vez que ocurre (1), o bien ocurre (2) o (4). $\mathcal{I}$ es un *modelo* de T si $\mathcal{I}$ satisface todos los axiomas en T.

Ejemplo 4.8.4 ((Heymans and Vermeir, 2002)) *Consideremos la siguiente ontología* $\Sigma = \langle T, \prec \rangle$ *donde:*

$$T \;=\; \left\{ \begin{array}{l} r_1 : \mathit{Thief} \sqsubseteq \mathit{Punished} \\ r_2 : \mathit{Minor} \sqsubseteq \neg \mathit{Punished} \\ r_3 : \mathit{Criminal} \sqsubseteq \mathit{Punished} \\ r_4 : \{\mathit{Bill}\} \sqsubseteq \mathit{Thief} \\ r_5 : \{\mathit{Bill}\} \sqsubseteq \mathit{Minor} \end{array} \right\}$$

con el orden $r_3 \prec r_2 \prec r_1$. *Si la interpretación de Bill es bill, tenemos dos modelos (ver Cuadro 4.2).*

El *soporte de un modelo* es el conjunto de los axiomas instanciados que son clásicamente satisfechos con respecto a un individuo x y una interpretación $\mathcal{I}$. Formalmente:

$$\mathcal{S}^{\mathcal{I}} = \left\{ \; (x, A \sqsubseteq B) : x \in \Delta^{\mathcal{I}} \text{ y } A \sqsubseteq B \text{ está clásicamente satisfecho con respecto a } x \; \right\}.$$

Un modelo $\mathcal{I}$ es *preferido* a otro modelo $\mathcal{J}$, denotado como $\mathcal{I} \leq \mathcal{J}$, si para todo $(x, A \sqsubseteq B) \in \mathcal{S}^{\mathcal{J}} \backslash \mathcal{S}^{\mathcal{I}}$ existe $(x, C \sqsubseteq D) \in \mathcal{S}^{\mathcal{I}} \backslash \mathcal{S}^{\mathcal{J}}$ tal que $(C \sqsubseteq D) \prec (A \sqsubseteq B)$. Intuitivamente, un modelo $\mathcal{I}$ es preferido a un modelo $\mathcal{J}$ si hay un axioma satisfecho por $\mathcal{J}$ y no por $\mathcal{I}$ que es derrotado por un axioma satisfecho por $\mathcal{I}$ y no por $\mathcal{J}$. Un modelo $\mathcal{I}$ es un *modelo preferido* si es minimal con respecto a la relación $\leq$ de comparación de modelos. En el caso del Ejemplo 4.8.4, la interpretación $\mathcal{I}_2$ será el modelo preferido, ya que es la que satisface clásicamente la mayor cantidad de axiomas.

	Thief	*Minor*	*Criminal*	*Punished*
$\mathcal{I}_1$	$\{bill\}$	$\{bill\}$	$\{bill\}$	$\{bill\}$
$\mathcal{I}_2$	$\{bill\}$	$\{bill\}$	$\emptyset$	$\emptyset$

Cuadro 4.2: Modelos a partir de la ontología $\langle T, \prec \rangle$

En la misma dirección que Heymans y Vermier, nosotros permitimos razonar a partir de ontologías inconsistentes. Sin embargo, hay ciertas diferencias en nuestros enfoques. Primero, en nuestro acercamiento, las ontologías DL son traducidas a DeLP mientras que en el enfoque de Heymans y Vermeir, el método de razonamiento es definido sobre el lenguaje de las DL. Segundo, Heymans y Vermeir solamente introducen un nivel de derrota; en cambio, en nuestro acercamiento, la derrota de un derrotador permite reinstaurar una afirmación previamente derrotada. Tercero, Heymans y Vermeir solamente usan precedencia de reglas para definir la noción de derrota; ya que el criterio de comparación de DeLP es modular, nuestro enfoque puede adaptarse al de ellos (via comparación de argumentos a través de precedencia de reglas) o utilizar un criterio de comparación más declarativo (via comparación de argumentos a través de especificidad generalizada).

Ejemplo 4.8.5 *La ontología presentada en el Ejemplo 4.8.4 puede ser representada como el programa DeLP:*

$$\Pi \;=\; \left\{ \begin{array}{l} thief(bill) \\ minor(bill) \end{array} \right\},\; y\; \Delta = \left\{ \begin{array}{l} r_1 : punished(X) \prec thief(X) \\ r_2 : \sim punished(X) \prec minor(X) \\ r_3 : punished(X) \prec criminal(X) \end{array} \right\}$$

con la precedencia de reglas $r_1 \prec r_2 \prec r_3$.[21] *De esta manera, tenemos dos argumentos* $\langle \mathcal{A}_1, punished(bill) \rangle$ *y* $\langle \mathcal{A}_2, \sim punished(bill) \rangle$*, con:*

$$\begin{array}{rcl} \mathcal{A}_1 & = & \left\{ \; punished(bill) \prec thief(bill) \; \right\}, y, \\ \mathcal{A}_2 & = & \left\{ \; \sim punished(bill) \prec minor(bill) \; \right\}. \end{array}$$

[21] Notar que el criterio de comparación de reglas de DeLP se nota exactamente al revés que en Heymans y Vermeir, pero que, sin embargo, el significado es exactamente el mismo. Es decir "$r_1 \prec r_2$" de acuerdo a la Definición 3.3.13 (página 77) quiere decir lo mismo que "$r_1 \succ r_2$" de acuerdo a la notación de Heymans y Vermeir.

En este contexto, usando como criterio de comparación de argumentos a la comparación de reglas definida en la Definición 3.3.4 (pág. 77), el argumento $\mathcal{A}_2$ derrota al argumento $\mathcal{A}_1$. En cambio, si supiéramos posteriormente que Bill es un criminal (i.e., $criminal(bill) \in \Pi$)*, tendríamos un argumento $\langle \mathcal{A}_3, punished(bill) \rangle$, con:*

$$\mathcal{A}_3 \;=\; \left\{\; punished(bill) \prec criminal(bill) \;\right\},$$

que derrotaría a $\mathcal{A}_2$ y reinstauraría a $\mathcal{A}_1$. De esta manera, nuestro sistema mostraría el mismo comportamiento que el exhibido por el de Heymans y Vermeir.

4.8.5. El acercamiento de Huang

De acuerdo a Zhisheng Huang *et al.*(Huang et al., 2004, 2005) hay dos maneras de lidiar con inconsistencias en ontologías. Una es diagnosticar y reparar la inconsistencia al encontrarla. El otro acercamiento consiste en simplemente evitar la inconsistencia y aplicar un método de razonamiento no estándar para obtener respuestas significativas. El acercamiento de Huang *et al.* (Huang et al., 2005, 2004) se enfoca en la segunda opción, ya que consideran que en el marco de la Web Semántica, uno importará muchas ontologías de diversas fuentes y la reparación de la mismas tiene una escala muy grande para que la reparación sea efectiva.

En presencia de una ontología inconsistente, dada una función de selección, la cual puede definirse basándose en relevancia sintáctica o semántica, seleccionan alguna subteoría consistente. Entonces, aplican un método estándar de razonamiento para hallar respuestas significativas. Si no se puede hallar una respuesta satisfactoria, entonces el grado de relevancia de la función de selección se hace menos restrictivo, de esta manera extendiendo la subteoría consistente para realizar el razonamiento.

4.8.6. Haase & Motik: Correspondencias entre ontologías

Para permitir la interoperabilidad entre aplicaciones en sistemas distribuidos de información basados en ontologías heterogéneas, es necesario definir formalmente la noción de un mapeo entre ontologías. En (Haase and Motik, 2005), Haase y Motik definen un sistema de mapeos para ontologías OWL-DL, donde los mapeos son expresadas como correspondencias entre consultas conjuntivas sobre ontologías. Como contestar consultas sobre tal sistema de mapeo general es no decidible, ellos identifican un fragmento decidible del sistema de mapeo, que corresponde a OWL-DL extendido con *reglas seguras respecto de DL*. También muestran cómo el sistema de mapeo puede ser aplicado a la tarea de la integración de ontologías y presentan un algoritmo para resolver consultas.

El lenguaje OWL provee soporte para expresar correspondencias entre los elementos de las ontologías. Sin embargo, este soporte es muy limitado para muchos propósitos prácticos pues solamente incluye subsunción y equivalencia entre clases y propiedades.

En el trabajo de Haase y Motik se sigue el marco de (Lenzerini, 2002) para formalizar la noción de correspondencia para ontologías OWL-DL, donde las mismas son expresadas como correspondencias entre consultas sobre ontologías. Sin embargo, como el caso general es indecidible, para obtener una alternativa más apropiada para apropiada para aplicaciones prácticas, introducen restricciones que permiten obtener decidibilidad. Estas correspondencias restringidas pueden ser expresadas en OWL-DL extendido con el llamado subconjunto *DL-safe* del Lenguaje de Reglas de la Web Semántica (en inglés, *Web Semantic Rule Language*) (Motik et al., 2004; Horrocks and Patel-Schneider, 2003).

Haase y Motik adaptan la definición de sistema de integración de ontologías de Calvanese (Calvanese et al., 2001) (véase la Sección 2.5) considerando a las ontologías representadas en OWL pero presentándolas con sintaxis DL; en su caso el sistema es llamado *sistema de correspondencias OWL-DL*. Basándose en la regla π de traducción de DL a Datalog conjuntivo, definen la semántica del sistema de integración de ontologías de una manera análoga a la presentada por nosotros en la Definición 4.6.3:

Definición 4.8.1 (Semántica del sistema de correspondencias (Haase and Motik, 2005)) *Para un sistemas de correspondencias $\mathcal{MS} = (\mathcal{S}, \mathcal{T}, \mathcal{M})$, sea $\pi(\mathcal{MS}) = \pi(\mathcal{S}) \cup \pi(\mathcal{T}) \cup \pi(\mathcal{M})$. La tarea de inferencia principal para $\mathcal{MS}$ es el cómputo de respuestas de $Q(x, y)$ con respecto a $\mathcal{MS}$, para una consulta conjuntiva $Q(x, y)$.*

Para evitar la no decidibilidad del caso general del sistema de correspondencias, Haase y Motik aplican una restricción a las correspondencias para convertir el problema de resolución de consultas en decidible. Su acercamiento es referido como *correspondencias seguras DL* (en inglés, *DL-safe mappings*). A continuación, resumimos brevemente tal acercamiento. Considere un mapeo sensato $q_S \sqsubseteq q_T$ con la aserción $\forall x.q_T(x, y_T) \leftarrow q_S(x, y_S)$. para evitar la introducción de nuevos elementos en la interpretación, se prohibe el uso de variables no distinguidas en la consulta q_T (*i.e.*, se restringen las aserciones a la forma $\forall x.q_T(x) \leftarrow q_S(x, y_S)$. De esta manera, se limita la aplicabilidad de las reglas a individuos conocidos.

Ejemplo 4.8.6 (Tomado de (Haase and Motik, 2005)) *Consideremos un sistema de integración de ontologías $(\mathcal{S}, \mathcal{T}, \mathcal{M})$, en el Cuadro 4.3 se presentan las ontologías fuente $\mathcal{S}$ y blanco $\mathcal{T}$ mientras que en el Cuadro 4.3 se presentan las correspondencias $\mathcal{M}$. Los elementos (nombres de conceptos y roles) de las ontologías $\mathcal{S}$ y $\mathcal{T}$ se hallan calificados con los nombres de espacios ("s" y "t") respectivamente. En dicho ejemplo se integran dos ontologías sobre publicaciones. La ontología fuente $\mathcal{S}$ define personas y publicaciones, los artículos y las tesis son publicaciones; una persona y un tópico tienen como atributo un nombre de tipo string; un autor es una persona, y una publicación tiene un título de tipo string y es acerca de un cierto tópico. Por otro lado, la ontología blanco $\mathcal{T}$ define los conceptos autor, entrada, artículo tesis de maestría y de doctorado; expresa que un*

Cuadro 4.3: Ontología fuente $\mathcal{S}$ y blanco $\mathcal{T}$

Ontología fuente $\mathcal{S}$	Ontología blanco $\mathcal{B}$
$Person \sqsubseteq \top$	$Author \sqsubseteq \top$
$Publication \sqsubseteq \top$	$Entry \sqsubseteq \top$
$Article \sqsubseteq Publication$	$Article \sqsubseteq Entry$
	$PhDThesis \sqsubseteq Entry$
$Topic \sqsubseteq \top$	
$Person \sqsubseteq \forall name.String$	$Author \sqsubseteq \forall name.String$
$Topic \sqsubseteq \forall name.String$	
$\top \sqsubseteq \forall author.Person$	$\top \sqsubseteq \forall author.Author$
$Publication \sqsubseteq \forall title.String$	$Entry \sqsubseteq \forall title.String$
$Publication \sqsubseteq \forall isAbout.Topic$	$Entry \sqsubseteq \forall subject.String$

Cuadro 4.4: Correspondencia $\mathcal{M}$

Correspondencias
$Q_{s,1}(x,y) : s : Publication(x) \wedge s : title(x,y)$ $Q_{t,1}(x,y) : t : Entry(x) \wedge t : title(x,y)$ $m_1 : Q_{s,1} \sqsubseteq Q_{t,1}$
$Q_{s,2}(x) : s : Article(x)$ $Q_{t,2}(x) : t : Article(x)$ $m_2 : Q_{s,2} \sqsubseteq Q_{t,2}$
$Q_{s,3}(x) : s : Thesis(x)$ $Q_{t,3}(x) : (t : MasterThesis(x) \sqcup t : PhDThesis)(x)$ $m_3 : Q_{s,3} \sqsubseteq Q_{t,3}$
$Q_{s,4}(x,y) : s : author(x,y)$ $Q_{t,4}(x,y) : t : author(x,y)$ $m_4 : Q_{s,4} \sqsubseteq Q_{t,4}$
$Q_{s,5}(x) : s : Person(x) \wedge s : author(y,x)$ $Q_{t,5}(x) : t : Author(x)$ $m_5 : Q_{s,5} \sqsubseteq Q_{t,5}$
$Q_{s,6}(x,z) : s : Publication(x) \wedge s : isAbout(x,y) \wedge s : name(y,z)$ $Q_{t,6}(x,z) : t : Entry(x) \wedge t : subject(x,z)$ $m_6 : Q_{s,6} \sqsubseteq Q_{t,6}$

artículo es una entrada, las tesis de maestría y doctorado son entradas, los autores tienen un atributo llamado nombre de tipo string, el rol autor tiene como rango a individuos de tipo autor, y las entradas tienen como atributos un título y un objeto ambos de tipo string.

Vemos que el ejemplo presentado por Haase y Motik puede ser trivialmente adaptado al sistema de integración de ontologías presentado en la Sección 4.6.

En resumen, el acercamiento de Haase y Motik presenta una formalización de un sistema general de correspondencias para ontologías OWL-DL. En este sistema de mapeos entre ontologías fuente y destino, las correspondencias son expresadas como consultas conjuntivas entre las ontologías. La expresividad del sistema de mapeos está expresado en el lenguaje de ontologías $\mathcal{SHOIN}(\mathcal{D})$, el lenguaje consultas conjuntivas y la flexibilidad

de las aserciones del acercamiento GLAV. El esquema de etiquetado de ontologías coincide con el utilizado por nosotros. Una de las ventajas que posee el acercamiento de Haase y Motik en comparación con el definido por nosotros es que el mismo permite la integración *global-as-view* pero también la *local-as-view* en contraposición a nuestro acercamiento que sólo permite la integración *global-as-view*. Sin embargo, la ventaja que tiene nuestro acercamiento es que el mismo es capaz de lidiar con ontologías inconsistentes pues el sistema de Haase y Motik es incapaz de hacerlo ya que su semántica está basada en la de Datalog.

4.8.7. Antoniou *et al.*: DR-PROLOG

Antoniou *et al.* (Antoniou et al., 2004*a*) proponen un sistema de razonamiento rebatible para la web semántica basado en su propio formalismo de razonamiento rebatible, llamado *Lógica Rebatible*. Acertadamente, Antoniou *et al.* notan que en la integración de ontologías, las inconsistencias surgen naturalmente, y en dicho marco, la lógica rebatible es muy útil para resolverlas. Su sistema es sintácticamente compatible con RuleML (RuleML, 2005), tiene reglas rebatibles y estrictas, está basado en una traducción a la programación en lógica con una sintaxis declarativa.

Básicamente, en el trabajo de (Antoniou and Bikakis, 2007), esta situación se puede caracterizar como *conflictos entre reglas* (Antoniou and Bikakis, 2007), los cuales se dan en los niveles de la capa de ontologías y en la capa de lógica y razonamiento. A nivel de la capa de ontologías, se tiene: herencia *default* entre ontologías, mezcla de ontologías; y a nivel de la capa de lógica y razonamiento, se tiene: reglas con excepciones como una manera natural de representar reglas de negocios, y razonamiento con información incompleta.

Antoniou and Bikakis (2007) reportan la implementación de un sistema de razonamiento rebatible para razonar sobre la Web. Sus principales características son las siguientes:

- Su interfaz es compatible con RuleML, considerado por ellos como el mayor esfuerzo de estandarización de reglas en la Web Semántica.
- DR-PROLOG está basado en Prolog. El núcleo del sistema consiste de una traducción bien estudiada de conocimiento rebatible en programas lógicos bajo la Semántica Bien Fundada (en inglés, *Well-Founded Semantics*).
- El foco principal está en la flexibilidad: Reglas estrictas y rebatibles, y prioridades son parte de la interfaz y de la implementación.
- El sistema puede razonar con reglas y conocimiento ontológico escrito en RDF Schema u OWL. Lo último ocurre a través de la transformación de construcciones RDFS y muchas de OWL en reglas.

Como resultado de esto, DR-PROLOG es un sistema declarativo que soporta:

- reglas, hechos y ontologías;
- todos los mayores estándares de la Web Semántica: RDF, RDFS, OWL, RuleML;
- reglas monotónicas y no-monotónicas, razonamiento con inconsistencias, hipótesis de mundo abierto y cerrado.

Para soportar el razonamiento con ontologías RDF/S y OWL, (Antoniou and Bikakis, 2007) traducen datos RDF a hechos lógicos, y sentencias RDFS y OWL como hechos lógicos y reglas. Para los datos RDF, el parser RDF de SWI-Prolog (SWI-Prolog, n.d.) es usado para transformar RDF en un formato intermedio, representando triplas como rdf(Subject, Predicate, Object). Cierto procesamiento adicional transforma los hechos en el formato Predicate(Subject, Object), y *corta* los espacios de nombres y los elementos "comentario" de los archivos RDF, excepto para los recursos que se refieren a RDF Schema u OWL, para los cuales la información de espacios de nombres es retenida.

Además, para procesar información expresada en RDF Schema, las siguientes reglas para capturar la semántica de RDF Schema son creadas:

```
a: C(X) :- rdf:type(X,C).
b: C(X) :- rdfs:subClassOf(Sc,C),Sc(X).
c: P(X,Y) :- rdfs:subPropertyOf(Sp,P),Sp(X,Y).
d: D(X) :- rdfs:domain(P,D),P(X,Z).
e: R(Z) :- rdfs:range(P,R),P(X,Z).
```

Partes de ontologías OWL pueden ser también traducidas usando reglas lógicas, las cuales capturan la semántica de algunos de los construcciones OWL. Por ejemplo, las siguientes reglas son usadas para capturar la semántica de propiedades transitivas, simétricas e inversas respectivamente:

```
o1: P(X,Z) :- P(X,Y), P(Y,Z), rdf:type(P,owl:TransitiveProperty).
o2: P(X,Y) :- P(Y,X), rdf:type(P,owl:SymmetricProperty).
o3: P(X,Y) :- Q(Y,X), owl:Inverseof(P,Q).
o4: Q(X,Y) :- P(Y,X), owl:Inverseof(P,Q).
```

Otras construcciones OWL las cuales también son soportadas por DR-PROLOG son: owl:equivalentClass, owl:equivalentProperty, owl:sameIndividualAs, owl:sameAs, restricciones de propiedad (owl:allValuesFrom, owl:hasValue), colecciones OWL. Todas las reglas anteriores son creadas en tiempo de compilación, *i.e.*, antes de que las consultas propiamente dichas tomen lugar.

En acuerdo con la filosofía de la programación en lógica, DR-PROLOG está diseñado para contestar consultas. Hay dos tipos de pruebas, dependiendo de la fuerza de la prueba en la que se esté interesado: demostrabilidad *definida* o *rebatible*. Además, (Antoniou and Bikakis, 2007) reporta el desarrollo de un DTD para representar teorías rebatibles que extiende los DTDs de RuleML.

Seguidamente analizaremos las semejanzas y diferencias del trabajo de Antoniou *et al.*con nuestro acercamiento. El acercamiento de (Antoniou and Bikakis, 2007) permite lidiar con inconsistencias en la Web Semántica como lo hace nuestro acercamiento. Su acercamiento distingue reglas rebatibles de estrictas como el nuestro y posee flexibilidad a la hora de definir distintas prioridades entre diferentes argumentos. De la misma manera, nuestro sistema, al estar basado en DeLP, permite reemplazar el criterio de comparación de argumentos en forma modular. Al traducir construcción RDFS y OWL a Prolog, un número de construcciones OWL no pueden ser capturadas por el poder expresivo de los lenguajes de reglas; en forma semejante, DeLP no es lo suficientemente expresivo para expresar ciertas construcciones de las Lógicas para la Descripción. De la misma manera que nuestro acercamiento basado en DeLP, DR-PROLOG está construido con la filosofía de contestar consultas. La extensión de los DTD de la iniciativa RuleML coincide en funcionalidad con nuestro lenguaje *X-DeLP* para el intercambio de programas rebatibles reportado con anterioridad cronológica en los trabajos (Gómez et al., 2005*a*,*c*) (véase la Sección 6.4). El acercamiento de (Antoniou and Bikakis, 2007) está basado en la Lógica Rebatible la cual tiene semántica declarativa basada en Prolog con semántica bien fundada; nuestro sistema, está basado en la Programación en Lógica Rebatible, la cual tiene una semántica procedural. Su sistema razona con reglas y conocimiento ontológico escrito en RDF Schema u OWL, el cual es traducido directamente en reglas Prolog; en cambio, nosotros, basamos nuestro acercamiento en la traducción de ontologías expresadas en Lógicas para la Descripción a DeLP y dejamos implícito que es posible expresar ontologías OWL en Lógicas para la Descripción.

4.8.8. El acercamiento Williams y Hunter

En (Williams and Hunter, 2007), Williams y Hunter introducen el *Marco Argumentativo basado en Ontologías* (en inglés, *Ontology-based Argumentation Framework* u OAF) que enlaza un formalismo lógico basado en argumentación con ontologías expresadas en Lógicas para la Descripción. Muestran cómo estos dos formalismos pueden ser acoplados al observar unas pocas restricciones y proveen características no provistas en ninguno de los formalismos por separado. Su trabajo es evaluado en el marco de un caso de estudio en toma de decisiones en la elección del tipo de tratamiento en el cáncer de mama, donde las reglas desarrolladas a partir de los resultados publicados de ensayos clínicos. Los autores muestran cómo OAF provee cinco ventajas: (1) facilita un uso claro de definicio-

nes entre múltiples autores; (2) permite relacionar términos en las ontologías y las reglas con aquellos específicos de la literatura del dominio del problema; (3) provee un ajuste entre la estructura de las pruebas clínicas y la estructura de sus reglas; (4) permite una economía significativa en el tamaño de la base de reglas en comparación con otros acercamientos actuales, y (5) les permite tomar ventaja de los desarrollos en los acercamiento argumentativos y ontológicos.

En esta sección presentaremos resumidamente los conceptos fundamentales definidos en el marco de Williams y Hunter.

Definición 4.8.2 (Marco argumentativo basado en ontologías (OAF) (Williams and Hunter, 2007)) *Un marco argumentativo basado en ontologías (OAF)* Ω_Δ*, para una cierta ontología DL* $\Delta = (\Delta_T, \Delta_A)$ *(*Δ_T *es la Tbox y* Δ_A *la Abox) es un conjunto finito de reglas rebatibles* Θ *y un conjunto de fórmulas fijas (hechos)* Γ*. Un OAF* Ω_Σ *es simplemente* $\Theta \cup \mathcal{F}$ *pero a veces puede denotarse por* (Θ, Γ)*.*

El programa rebatible está compuesto por reglas rebatibles y hechos. Los hechos son obtenidos a partir de la Abox de la ontología. La manera de relacionar el programa rebatible con la Tbox de la ontología es a través de los predicados especiales $\texttt{conflict}(\phi, \psi)$ y $\texttt{entails}(\phi, \psi)$ que sirven para acceder a la ontología subyacente desde las reglas del programa rebatible. Eso se realiza a través de una función de *roll-up* de consultas (Horrocks and Tessaris, 2000) denotada como ρ.

Ejemplo 4.8.7 (Tomado de (Williams and Hunter, 2007)) *Consideremos la fórmula* ϕ *con* $\phi = \mathsf{People}(x) \wedge \mathsf{hasDisease}(x, y) \wedge \mathsf{BreastCancer}(y)$*, entonces* $\rho(\phi) = \mathsf{People}(x) \sqcap \exists \mathsf{hasDisease}.\mathsf{BreastCancer}$.

Williams y Hunter dan dos nociones propias de *deducción* (en inglés, *entailment*) y conflicto en el contexto de su sistema. Intuitivamente, una fórmula ψ se deduce de otra fórmula ϕ si ψ se deduce de ϕ en la ontología subyacente. Una fórmula consistente ϕ está en conflicto con otra fórmula consistente ψ si la aceptación simultánea de ambas fórmulas produce una inconsistencia en la ontología subyacente.

Definición 4.8.3 (Deducción. Conflicto (Williams and Hunter, 2007)) *Para dos fórmulas* $\phi, \psi \in \mathcal{F}$*,* $\texttt{entails}(\phi, \psi)$ *si y sólo si* $\Delta_T \cup \rho(\phi) \vdash_{Ont} \phi(\psi)$*.* $\texttt{conflict}(\phi, \psi)$ *si y sólo si* $\Delta_T \cup (\rho(\psi) \sqcap \rho(\psi)) \vdash_{Ont} \bot$ *y* $\Delta_T \cup \rho(\phi) \nvdash_{Ont} \top \sqsubseteq \bot$ *ni* $\Delta_T \cup \rho(\psi) \nvdash_{Ont} \top \sqsubseteq \bot$*.*

En general, $\texttt{entails}(\phi, \psi)$ vale en dos circunstancias.: la primera es cuando ϕ es una subclase de ψ y la segunda es cuando ϕ es equivalente a ψ. También hay que notar que la noción de conflicto es simétrica.

Las nociones de derivación (y por lo tanto de argumento) y de contradicción son una adaptación de las definiciones propias de DeLP (véase la Sección 3.3) pero en lugar de definirse solamente con respecto a un conjunto de reglas además se lo hace con respecto

a la ontología subyacente. Una *derivación rebatible* de ϕ a partir de $\Omega_\Delta = (\Theta, \Gamma)$ consiste de una secuencia finita $\phi_1, \ldots, \phi_n$ de fórmulas fijas, y cada fórmula está en la secuencia porque ϕ_i pertenece a Γ o existe una regla $r \in \Theta$ cuya cabeza es ϕ_i y el cuerpo es $\phi_1, \ldots, \phi_k$ y cada literal del cuerpo es o bien un elemento ϕ_j de la secuencia apareciendo antes que ϕ_i ($j < i$), o cada literal del cuerpo se deduce de un elemento ϕ_j de la secuencia tal que $\texttt{entails}(\phi_j, \phi_i)$ o se deduce de algún ϕ_j en Γ. En la definición de derivación rebatible vemos una diferencia grande con DeLP en el sentido en que una nueva información rebatible puede deducirse usando una regla rebatible o bien a partir de la ontología usando como premisas conclusiones rebatibles derivadas previamente.

Un conjunto de reglas es Θ es contradictorio si y sólo si existe una derivación rebatible para fórmulas ϕ_1 y ϕ_2 a partir de (Θ, Γ) tal que vale $\texttt{conflict}(\phi_1, \phi_2)$. Un argumento $Label : \langle A, \phi \rangle$ es un conjunto de reglas rebatibles minimal y no contradictorio obtenido a partir de una derivación de ϕ. Un argumento $\langle B, \psi \rangle$ es un *subargumento* de $\langle A, \phi \rangle$ si $B \subseteq A$.

Dos fórmulas ϕ y ϕ_1 en Ω están en *desacuerdo* si y sólo si vale que $\texttt{conflict}(\phi, \phi_1)$. Las definiciones para la relación de preferencia entre argumentos, derrotadores propios y por bloqueo, línea de argumentación, argumentos de soporte e interferencia, concordancia, línea de argumentación aceptable, árbol de dialéctica y garantía presentadas en (Williams and Hunter, 2007) son equivalentes a las definidas en DeLP por (García and Simari, 2004) (véase la Sección 3.3).

Ejemplo 4.8.8 (Basado en (Williams and Hunter, 2007)) *Un ejemplo de una regla como las utilizadas por Williams y Hunter es como la siguiente:*

$$\begin{aligned} r_a \quad : \quad & \mathsf{People}(x) \wedge \mathsf{hasDisease}(x, y) \wedge \mathsf{BreastCancer}(y) \\ & \wedge \mathsf{Tamoxifen5Yr}(z) \Rightarrow \mathsf{hasTreatment}(x, z) \end{aligned}$$

Esta regla dice que las personas que sufren de cáncer de mama pueden ser tratados con la droga Tamoxifen5Yr.

En resumen, Wiliams y Hunter (Williams and Hunter, 2007) usan argumentación para razonar con conjuntos de reglas posiblemente inconsistentes definidas sobre ontologías DL. La novedad del trabajo Wiliams y Hunter reside en que la información derivada para la construcción de argumentos puede obtenerse a partir de las reglas y a partir de la información presente en la ontología.

En contraste, nosotros traducimos ontologías posiblemente inconsistentes a DeLP para razonar con ellas dentro del marco de DeLP.

4.8.9. Acuerdo entre correspondencias entre ontologías: Laera *et al.*

Para soportar la interoperación semántica en ambientes abiertos, donde los agentes pueden sumarse o ausentarse y no es posible realizar ninguna suposición sobre las ontologías a alinear, los agentes involucrados necesitan acordar la semántica de los términos durante la interoperación. Llegar a este acuerdo puede solamente ser realizado a través de cierto tipo de negociación. Verdaderamente, los agentes diferirán en las ontologías del dominio con las que se comprometen, y, por lo tanto, en su percepción del mundo y en la elección del vocabulario que utilizan para representar conceptos.

En (Laera et al., 2006, 2007), Laera *et al.* proponen un acercamiento para soportar la creación e intercambio de diferentes argumentos, que soportan o rechazan posibles correspondencias. Cada agente puede decidir, de acuerdo a sus preferencias, si acepta o rechaza una correspondencia candidata. El marco propuesto considera argumentos y proposiciones específicos a la tarea de combinación (en inglés, *matching*) y están basados en la semántica de la ontología. Este marco de argumentación se basa en un esquema de manipulación de argumentos y en una codificación de las preferencias de los agentes con respecto a tipos particulares de argumentos.

Un *marco de argumentación* es un par $AF = \langle AR, A\rangle$ donde AR es un conjunto de argumentos y $A \subset AR \times AR$ es la relación de *ataque* para AF, y está compuesta por pares ordenados de distintos argumentos en AR. Un par $\langle x, y\rangle$ es referido como "x *ataca* a y", mientras que un conjunto de argumentos S se dice que ataca a un argumento y si y es atacado por un argumento en S.

Un *marco de argumentación basado en valores* (VAF) se define como $\langle AR, A, \nu, \eta\rangle$, donde $\langle AR, A\rangle$ es un marco de argumentación, ν es un conjunto de k valores que representan los tipos de argumentos y $\eta : AR \rightarrow \nu$ es una correspondencia que asocia un valor $\eta(x) \in \nu$ a cada argumento $x \in AR$.

Una *audiencia* para un VAF es una relación binaria $\mathcal{R} \subseteq \nu \times \nu$ cuya clausura transitiva (irreflexiva) $\mathcal{R}^*$ es asimétrica, *i.e.* a lo sumo uno de (v, v'), (v', v) son miembros de $\mathcal{R}^*$ para cualquier $v, v' \in \nu$ distintos. Se dirá que v_i *es preferido a* v_j *en la audiencia* $\mathcal{R}$, denotado $v_i \prec_R v_j$, si $(v_i, v_j) \in \mathcal{R}^*$. Sea $\mathcal{R}$ una audiencia, α es una *audiencia específica* (compatible con $\mathcal{R}$) si α es un *orden total* de ν y $\forall v, v' \in \nu, (v, v') \in \alpha \Rightarrow (v', v) \notin \mathcal{R}^*$.

Sea $\langle AR, A, \nu, \eta\rangle$ un VAF y $\mathcal{R}$ una audiencia. Para argumentos x, y en AR, x es un *ataque exitoso a* y *con respecto a la audiencia* $\mathcal{R}$ si $(x, y) \in A$ y no ocurre que $\eta(y) \succ_{\mathcal{R}} \eta(x)$. Un argumento x es *aceptable al subconjunto* S con respecto a una audiencia $\mathcal{R}$ si: por cada $y \in AR$ que exitosamente ataca a x con respecto a $\mathcal{R}$, hay un $z \in D$ que satisfactoriamente ataca a y con respecto a $\mathcal{R}$. Un subconjunto S de AR es *libre de conflicto con respecto a la audiencia* $\mathcal{R}$ si: por cada $(x, y) \in S \times S$, o bien $(x, y) \notin A$ o $\eta(y) \succ_{\mathcal{R}} \eta(x)$. Un subconjunto S de AR es *admisible* con respecto a la audiencia $\mathcal{R}$ si

S es libre de conflicto con respecto a $\mathcal{R}$ y cada $x \in S$ es aceptable a S con respecto a $\mathcal{R}$. Un subconjunto S es una *extensión preferida* para la audiencia $\mathcal{R}$ si es un conjunto admisible maximal con respecto a $\mathcal{R}$. Intuitivamente, la noción de extensión preferida representa una posición consistente dentro de AF, la cual es posible defendible contra todos los ataques la cual no puede ser extendida sin volverse inconsistente o abierta al ataque.

Dado un VAF $\langle AR, A, \nu, \eta \rangle$, un argumento $x \in AR$ es *subjetivamente aceptable* si y sólo si x aparece en la extensión preferida para algunas audiencias específicas pero no en todas. Un argumento $x \in AR$ es *objetivamente aceptable* si y sólo si x aparece en la extensión preferida de *cada* audiencia específica. Un argumento que no ni subjetiva u objetivamente aceptable es llamado *indefendible*. Un argumento $x \in AF$ es una tripla $x = \langle G, m, \sigma \rangle$ donde m es una correspondencia $\langle e, e', n, R \rangle$; G son las premisas justificando una creencia prima facie indicando si la correspondencia vale o no vale, y σ es una de $\{+, -\}$ dependiendo de si el argumento indica que m vale o no. Un argumento $y \in AF$ *derrota* a un argumento $x \in AF$ si x e y son argumentos para el mismo mapeo pero con diferentes signos, *e.g.* si x e y son de la forma $x = \langle G_1, m, + \rangle$ y $x = \langle G_2, m, - \rangle$, x *contraargumenta* a y y viceversa.

Un agente Ag_i es caracterizado por una 4-upla $\langle O_i, VAF_i, Pref_i, \epsilon_i \rangle$ donde O_i es una ontología OWL; $VAF_i = \langle AR_i, A_i, \nu, \eta_i$ es el VAF del agente Ag_i; $Pref_i$ es el preorden privado de las preferencias sobre ν y ϵ_i es el valor privado de umbral. Un conjunto de agentes $A = \{Ag_1, \ldots, Ag_n\}$ forma un sistema multiagente. El conjunto de todos los argumentos compartidos por todos los agentes no son necesariamente disjuntos, mientras los valores ν son comunes y compartidos por todos los agentes.

Las preferencias y los umbrales seleccionados por un agente dependen de su contexto y situación. Una característica en este contexto es la ontología del agente, y sus características estructurales, tales como la profundidad de su jerarquía de clases y factor de ramificación, tasa de propiedades a conceptos, etc. Así un agente puede determinar sus preferencias y umbrales en función de métricas definidas sobre su ontología. Si se asumen n agentes, comprometiéndose a diferentes ontologías, y un repositorio de mapeos OMR almacenando los conjuntos de correspondencias entre las ontologías de los agentes (generadas por servicios de mapeo). Ya que diferentes conjuntos de mapeos pueden ser generados por estos servicios, y para permitir que los agentes lleguen a un acuerdo para el servicio requerido sin llegar a una alineación completa entre las ontologías, el agente peticionante especifica cuáles de las componentes de su propia ontología están involucradas en el requerimiento del servicio, y sólo busca generar una alineación cuando con la ontología del otro agente con respecto a estas entidades.

Dado un agente Ag_i y un mapeo candidato m_i con un conjunto de justificaciones G, el agente primero evalúa la aceptabilidad de m, este mapeo será aceptado si el grado de credibilidad del mapeo es superior al umbral privado del agente. Consecuentemente

el agente genera un conjunto de argumentos $x = \langle G, m, + \rangle$ instanciando el esquema argumentativo. En caso contrario, el agente rechaza el mapeo y genera argumentos $x = \langle G, m, - \rangle$ en contra del mismo. La aceptación o rechazo de un mapeo se realiza a través de un protocolo en el cual n ($n > 2$) agentes argumentan acerca de la aceptabilidad del mapeo potencial, arribando a una solución que está basada en sus preferencias y la información que intercambian durante la argumentación. Formalmente, el protocolo de argumentación de (Laera et al., 2006) es una n-upla $\langle$ *Mappings, Agents, Acts, Replies, Move, Dialogue, Result* $\rangle$ tal que: *Mapping* es un mapeo candidato en OMR sujeto a evaluación entre los agentes; *Agents* es el conjunto de agentes $Ag_1, \ldots, Ag_n$ tomando parte en el diálogo; *Acts* es el conjunto de posibles actos de argumentación tal que *Acts* = { *Support, Contest, Withdraw* }; *Replies* es un mapeo de *Acts* en *Acts* que asocia a cada acto su posible respuesta; *Move* es una tupla indicando la movida del agente en el proceso de negociación; *Dialogue* es una secuencia finita no vacía de movidas, y, *Result* es una función de *Mapping* en el conjunto { *agreed, agreeable, rejected* }. En el proceso de diálogo, cada agente evalúa la aceptabilidad del mapeo m y entonces genera un conjunto de argumentos a favor del mismo o un conjunto de contraargumentos en contra del mismo. Si el agente no tiene argumentos a favor o en contra, entonces se retira con el acto $Widthdraw(m)$ y el diálogo termina. Cada agente Ag_i verifica si está de acuerdo con el conjunto de mapeos $m_1, \ldots, m_k$ usado en los argumentos que intercambiaron. Si tienen argumentos o contraargumentos que presentar, empiezan un nuevo diálogo para evaluar cada uno de estos mapeos. El diálogo termina cuando cada mapeo involucrado en el diálogo inicial sobre m ha sido evaluado o el agente se retira. Cuando el diálogo termina, cada agente construye el marco de argumentación, y corrobora la aceptabilidad del mapeo por medio del cómputo de sus extensiones preferidas.

4.8.10. El acercamiento de J. van Diggelen

Van Diggelen (van Diggelen, 2007) ataca el problema de la interoperabilidad entre agentes en el contexto de un sistema multiagente. Van Diggelen argumenta que una ontología es usualmente insuficiente para proveer la interoperabilidad semántica entre agentes. En cambio, cada agente conoce sólamente su propia ontología y no conoce la de los otros agentes. Su acercamiento, basado en diálogos, propone aumentar las ontologías locales de cada agente sólo con la información que necesitan saber para poder interactuar con los otros agentes del MAS.

Desde el punto de vista de van Diggelen *et al.*, la comunicación en Sistemas Multi-Agentes (MAS) heterogéneos está limitada por la falta de ontologías compartidas. La negociación de ontologías ofrece un acercamiento integrado que permite a los agentes gradualmente aportar a la construcción de un sistema integrado compartiendo parte de sus ontologías. Esta solución involucra la combinación de protocolo normal de comunicación

entre agentes junto con un protocolo de alineación de ontologías.

La comunicación entre agentes involucrando negociación de ontologías es establecido por dos protocolos: el *protocolo normal de comunicaciones* y el *protocolo de alineación de ontologías*. El objetivo del protocolo normal de comunicación es pasar conocimiento asercional (*i.e.*, conocimiento relevante a una tarea o problema particular). La comunicación normal procede haciendo uso de una ontología intermedia compartida, la cual indirectamente alínea las ontologías locales de los agentes. Como la ontología intermedia es usada sólo para propósitos de comunicación, van Diggelen *et al.*se refieren a la misma como *vocabulario de comunicación*. Los agentes participan en la comunicación normal traduciendo desde y hacia el vocabulario de comunicación. Cuando el vocabulario de comunicación alínea insuficientemente las ontologías locales para permitir la comunicación normal, los agentes hacen una transición al protocolo de alineación de ontologías. Este protocolo tiene el objetivo de restablecer la comunicación normal agregando conceptos al vocabulario de comunicación. Esto es realizado usando *intercambio de ontologías*: un agente le enseña información ontológica a otro agente. Luego de que los agentes han resuelto el problema, retornan al protocolo normal de comunicación.

El enfoque descripto es caracterizado por medio de *negociación de ontologías*. El proceso de negociación viene dado porque para cada agente es más barato que los demás agentes *adapten* sus respectivas ontologías locales en lugar de *aprender* las ontologías de los demás agentes.

Es de notar que en el acercamiento utilizado en esta tesis para integrar ontologías las reglas de articulación entre ontologías son dadas y no son computadas. De esta manera, el acercamiento de van Diggelen puede ser considerado complementario al presentado en esta tesis.

4.8.11. Relación entre los acercamientos presentados

Grosof *et al.* Grosof et al. (2003) muestran cómo interoperar, semántica e inferencialmente, entre los acercamientos principales a la Web Semántica a las reglas (Programas Lógicos RuleML) y las ontologías (OWL DL) para analizar su intersección expresiva. Definen un nuevo lenguaje intermedio de representación de conocimiento llamado Programas Lógicos para la Descripción (en inglés, *Description Logic Programs (DLP)*), y la cercana Lógica de Horn para la Descripción (en inglés, *Description Horn Logic (DHL)*) la cual es un fragmento expresivo de la Lógica de Primer Orden. Muestran cómo realizar la traducción de premisas e inferencias del fragmento DLP de las DL a la programación en lógica. Parte de nuestro acercamiento está basado en el trabajo de Grosof ya que el algoritmo de traducción de ontologías DL hacia DeLP está basado en él. Sin embargo, como Grosof utiliza reglas Prolog estándar, no son capaces de lidiar con bases de conocimiento inconsistentes como sí lo hace nuestra propuesta.

Heymans and Vermeir (2002) extienden la DL $\mathcal{SHOQ}(D)$ con un orden de preferencia en los axiomas. Con este orden parcial estricto ciertos axiomas pueden ser denegados si son derrotados por otros axiomas más preferidos. También imponen una semántica de modelos preferidos, introduciendo nomonotonicidad en $\mathcal{SHOQ}(D)$. Similarmente a Heymans and Vermeir (2002), permitimos realizar inferencias a partir de ontologías inconsistentes al considerar subconjuntos (argumentos) de la ontología original. Heymans and Vermeir (2002) también imponen un criterio de comparación fijo (en inglés, *hardcoded*) sobre los axiomas DL. En nuestro trabajo, el sistema, y no el programador, decide cuáles axiomas serán preferidos ya que utilizamos al criterio de especificidad como criterio de comparación de argumentos. Pensamos que nuestro acercamiento puede ser considerado más declarativo en este respecto. En particular, como el criterio de comparación de argumentos de DeLP es modular, el criterio de comparación de reglas también puede ser adoptado (como se mostró en la Sección 4.8.4).

Eiter *et al.* Eiter et al. (2004) proponen una combinación de la programación en lógica bajo la semántica de conjuntos de respuestas (en inglés, *answer sets*) con las DLs $\mathcal{SHIF}(D)$ y $\mathcal{SHOIN}(D)$. Esta combinación permite construir reglas sobre las ontologías. En contraste con nuestro acercamiento, ellos mantienen separadas las reglas y las ontologías y manejan excepciones codificándolas explícitamente en programas bajo la semántica de conjuntos de respuestas.

En el acercamiento de Mitra (Mitra, 2004), las reglas de articulación entre ontologías son generadas en forma semi-automática mientras que en nuestro acercamiento asumimos las reglas de articulación como dadas. Ciertamente, el acercamiento de Mitra puede ser considerado complementario al propuesto en esta Tesis.

Huang *et al.* Huang et al. (2005) usan lógica paraconsistente para razonar con ontologías inconsistentes. Usan una función de selección para determinar cuáles subconjuntos consistentes de una ontología inconsistente deberían ser considerados en el proceso de razonamiento. En nuestro acercamiento, dada una ontología inconsistente $\Sigma = (T_S, T_D, A)$, consideramos el conjunto de argumentos garantizados a partir del programa DeLP $\mathcal{P} = (\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$ como las consecuencias válidas de Σ. Note que en DeLP, dado el programa $\mathcal{P}$, cada argumento obtenible a partir de $\mathcal{P}$ es consistente. Además, el conjunto de todos los argumentos garantizados con respecto a $\mathcal{P}$ es consistente.

Williams y Hunter Williams and Hunter (2007) usan argumentación para razonar con conjuntos de reglas posiblemente inconsistentes cuyo vocabulario se halla definido sobre ontologías DL. En constraste con nuestro acercamiento, nosotros traducimos ontologías DL posiblemente inconsistentes hacia DeLP para razonar con ellas dentro de DeLP.

Laera *et al.* Laera et al. (2006) proponen un acercamiento para soportar la creación e intercambio de diferentes argumentos, que soportan o rechazan posibles correspondencias entre ontologías en el contexto de un sistema multi-agente. En nuestro trabajo, cuando planteamos un sistema de integración de ontologías, asumimos las correspondencias entre

ontologías como dadas.

Antoniou *et al.* Antoniou and Bikakis (2007) proponen un acercamiento basado en reglas apoyado en la traducción hacia la programación en lógica con semántica declarativa capaz de razonar con reglas, RDF, RDF Schema y parte de ontologías OWL. En contraste con nuestro acercamiento, la argumentación no es usada explícitamente para razonar sobre ontologías inconsistentes. En cambio, se traduce sentencias OWL a Prolog para razonar con ellas con *Defeasible Logic*.

Ya que en el acercamiento utilizado en esta tesis para integrar ontologías las reglas de articulación entre ontologías son dadas y no son computadas. De esta manera, el acercamiento de Van Diggelen (van Diggelen, 2007) puede ser considerado complementario al presentado en esta tesis.

4.9. Resumen

Hemos presentado un acercamiento novedoso basado en la argumentación rebatible para el razonamiento en presencia de ontologías posiblemente inconsistentes. Como se discutió en la introducción, dada una ontología expresada en una Lógica para la Descripción, proponemos traducirla a un programa lógico rebatible. Luego, las consultas realizadas con respecto a la ontología original serán en cambio resueltas con respecto a un programa DeLP que representa el contenido de la ontología original.

Hemos mostrado cómo resolver ejemplos relevantes de la literatura, hemos aplicado este acercamiento al problema de la integración de ontologías y hemos mostrado como adaptar nuestro acercamiento a aquellos de los trabajos relacionados cuando esto fuera posible.

En el Capítulo 5 analizaremos las propiedades formales que se desprenden de este acercamiento novedoso al tratamiento de ontologías inconsistentes en la Web Semántica.

Capítulo 5

Propiedades del acercamiento

En este capítulo presentamos algunas de las propiedades formales que surgen del formalismo de las δ-ontologías presentado en el Capítulo 4. En primer lugar, consideraremos las propiedades del sistema de razonamiento con ontologías presentado en el capítulo anterior. Las propiedades emergentes de este formalismo están íntimamente relacionadas con las propiedades intrínsecas del formalismo de la Programación en Lógica Rebatible. Posteriormente, consideraremos las propiedades que surgen de establecer una relación entre la interpretación tradicional de las ontologías expresadas en Lógicas para la Descripción con respecto al marco de razonamiento establecido por las δ-ontologías.

5.1. Propiedades internas respecto de la dinámica del razonamiento argumentativo

5.1.1. Chequeo de instancia

Cuando una δ-ontología es interpretada como un programa DeLP, a cada aserción de la Abox corresponde un hecho en el programa DeLP que interpreta a tal δ-ontología. De esta observación, se desprenden varias propiedades relacionadas con la pertenencia a los conceptos involucrados de los individuos mencionados en aserciones pertenecientes a la Abox. Para llevar a cabo tal tarea, primero repasamos una propiedad mostrada por García and Simari (2004) que será útil para estudiar las propiedades de nuestro acercamiento que deseamos mostrar.

Lema 5.1.1 ((García and Simari (2004); Prop. 3.1)) *Dado un programa DeLP* $\mathcal{P} = (\Pi, \Delta)$*, la estructura de argumento* $\langle \emptyset, Q \rangle$ *no posee derrotadores.*

Demostración: *Por el absurdo, supongamos que existe un derrotador* $\langle \mathcal{A}, H \rangle$ *para* $\langle \emptyset, Q \rangle$*. Luego, los literales* Q *y* H *están en desacuerdo. Entonces,* $\Pi \cup \{H, Q\} \vdash \bot$*. Sin embargo,*

como H se deriva de $\Pi \cup \mathcal{A}$ y Q se deriva de Π, debe ser que $\Pi \cup \mathcal{A} \vdash \bot$; lo cual implica que $\langle \mathcal{A}, H \rangle$ no es un argumento válido. □

Entonces, utilizando la propiedad establecida por el Lema 5.1.1, enunciamos la siguiente propiedad:

Propiedad 5.1.1 *Sea $\Sigma = (T_S, T_D, A)$ una δ-ontología. Por cada aserción de concepto de la forma $a : C$ perteneciente a A se cumple que C_s^a.*

Demostración: *Sea $\mathcal{P} = (\Pi, \Delta)$ el programa DeLP que interpreta a Σ. Supongamos entonces que $\Pi = \mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A)$. Supongamos que $a : C$ pertenece a A. Por lo tanto, como $\mathfrak{T}_\Pi(a : C) = C(a)$, se da el caso que $C(a)$ pertenece a Π. Entonces $C(a)$ se deriva trivialmente en forma estricta a partir de $\mathcal{P}$.* □

Una observación interesante que se desprende de la propiedad anterior es la siguiente:

Propiedad 5.1.2 *Sea $\Sigma = (T_S, T_D, A)$ una δ-ontología. Por cada aserción de concepto de la forma $a : C$ perteneciente a A se cumple que C_j^a y C_p^a.*

Demostración: *Si $\mathcal{P} = (\Pi, \Delta) = (\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$, en virtud de la Propiedad 5.1.1, el individuo a pertenece estrictamente al concepto C ya que $C(a)$ pertenece a Π. Pero como $C(a)$ se deriva estrictamente a partir de $\mathcal{P}$, es el caso que existe una estructura de argumento $\langle \emptyset, C(a) \rangle$. Luego, el individuo a pertenece potencialmente al concepto C. Además, por el Lema 5.1.1, tal argumento no posee contraargumentos y, por lo tanto, está justificado. Así, el individuo a pertenece justificadamente al concepto C.* □

Una propiedad mostrada por García and Simari (2004) expresa que ningún literal derivado estrictamente a partir de un programa DeLP puede ser contraargumento para ningún argumento obtenido a partir de ese mismo programa. Formalmente:

Lema 5.1.2 *Dado un programa DeLP $\mathcal{P} = (\Pi, \Delta)$, $\langle \emptyset, Q \rangle$ no puede ser contraargumento para ningún argumento $\langle \mathcal{A}, H \rangle$.*

Demostración: *Si $\langle \emptyset, Q \rangle$ es un contraargumento de $\langle \mathcal{A}, H \rangle$, entonces es el caso que Q y H están en desacuerdo. Por lo tanto, $\Pi \cup \{Q, H\}$ permiten derivar literales complementarios. Por lo tanto, como H se deriva de $\Pi \cup \mathcal{A}$ y Q se deriva de Π, $\Pi \cup \mathcal{A}$ permite derivar literales complementarios. Luego, de acuerdo a la Definición 3.3.8, $\langle \mathcal{A}, H \rangle$ no es un argumento válido.* □

Esta propiedad nos lleva a definir la siguiente noción de *desacuerdo de pertenencia potencial* que nos va a permitir caracterizar más profundamente el vínculo entre las relaciones de pertenencias estrictas, justificada y potencial de individuos a clases.

Definición 5.1.1 (Desacuerdo de pertenencia potencial) *Sea Σ una δ-ontología. La pertenencia potencial de un individuo "a" a un concepto "C" está en desacuerdo con la*

pertenencia de un individuo "b" al concepto "D" con respecto a Σ si y sólo si los literales "$C(a)$" y "$D(b)$" están en desacuerdo con respecto al programa DeLP $\mathfrak{T}(\Sigma)$.

En base a esta definición podemos enunciar la siguiente propiedad:

Propiedad 5.1.3 *Sea Σ una δ-ontología. Sean a, b nombres de individuos en Σ y sean C, D nombres de conceptos en Σ. Si es el caso que C_s^a, no ocurre que D_p^b cuando la pertenencia potencial de a al concepto C esté en desacuerdo con la pertenencia de b al concepto D.*

Demostración: *Por el Lema 5.1.2, en DeLP vale que $\langle \emptyset, C(a)\rangle$ no puede ser contraargumento para ningún argumento $\langle \mathcal{A}, D(b)\rangle$ con respecto al programa DeLP $\mathfrak{T}(\Sigma)$. Luego, si ocurre que D_p^b y $C(a)$ y $D(b)$ están en desacuerdo, en realidad no puede ser el caso que D_p^b.* □

Así, si se dan estas dos condiciones, entonces no hay desacuerdo entre la pertenencia potencial de "b" a "D" y la pertenencia de "a" a "C". Un corolario que se desprende trivialmente de esta propiedad es que si "a" pertenece estrictamente a "C" y tal caso está en desacuerdo con la pertenencia potencial de "b" a "D", entonces tampoco puede ocurrir que "b" pertenezca justificadamente a "D". Esta situación se da trivialmente, ya que si "b" no puede pertenecer potencialmente a "D", tampoco lo puede hacer justificadamente.

Está claro que tampoco "b" puede pertenecer estrictamente a "D" puesto que en este caso la Abox sería inconsistentente respecto de la Tbox (véase la Definición 4.5.2 en la página 124). Si este fuera el caso, tendríamos dos argumentos $\langle \emptyset, C(a)\rangle$ y $\langle \emptyset, D(b)\rangle$ tal que $C(a)$ y $D(b)$ están en desacuerdo con respecto a $\mathfrak{T}(\Sigma) = (\Pi, \Delta)$. Luego, $\Pi \cup \{C(a), D(b)\}$ permite derivar literales complementarios. Por lo tanto Π es inconsistente.

Si el individuo a pertenece estrictamente al concepto C entonces a pertenece justificadamente a C; si el individuo a pertenece justificadamente al concepto C entonces a pertenece potencialmente a C. Formalmente:

Propiedad 5.1.4 *Sea Σ una δ-ontología, C un nombre de concepto definido en Σ y a un nombre de individuo. C_s^a implica C_j^a y C_j^a implica C_p^a.*

Demostración: *Sea $\mathcal{P} = (\Pi, \Delta)$ el programa DeLP que da semántica a Σ. Probemos primero la primera afirmación. Si vale que C_s^a entonces existe un argumento $\langle \emptyset, C(a)\rangle$ construido a partir de $\mathcal{P}$. Como en DeLP, en virtud del Lema 5.1.1, los argumentos vacíos no pueden ser atacados, están por lo tanto garantizados. Luego, vale que C_j^a.*

La segunda parte vale trivialmente porque los argumentos garantizados son también argumentos. Es decir, si ocurre que C_j^a, existe un argumento garantizado $\langle \mathcal{A}, C(a)\rangle$ con respecto a $\mathcal{P}$. Luego, trivialmente vale que C_p^a. □

Propiedad 5.1.5 *Dada una δ-ontología $\Sigma = (T_S, T_D, A)$, un nombre de concepto C y un nombre de individuo a. Si $a : C$ pertenece a A entonces vale que C_j^a y C_p^a.*

Demostración: *Si $a : C$ pertenece a A, entonces por la Propiedad 5.1.1, ocurre que C_s^a. Luego, por la Propiedad 5.1.4 vale que C_j^a y C_p^a.* □

Debido a que las Lógicas para la Descripción son un subconjunto de la Lógica de Primer Orden, en el acercamiento tradicional (en el sentido de (Baader et al., 2003)) con ontologías expresadas en DL, la inconsistencia de una ontología tiene un efecto *explosivo*. Es decir, a partir de una ontología inconsistente es posible demostrar cualquier afirmación. En cambio, estamos interesados en que en el formalismo de las δ-ontologías, las inferencias realizadas posean cierta coherencia. En particular, no es posible arribar a conclusiones contradictorias a partir de una δ-ontología. Las siguientes propiedades caracterizan formalmente estas nociones.

Primero, repasemos la propiedad que dice que en DeLP no es posible garantizar un literal y su complemento al mismo tiempo; esta propiedad será posteriormente útil para demostrar ciertas propiedades de nuestro formalismo de razonamiento con ontologías.

Propiedad 5.1.6 *Sea $\mathcal{P} = (\Pi, \Delta)$ un programa DeLP. Sea L un literal. No es posible garantizar L y $\overline{L}$ al mismo tiempo.*

Demostración: *Por el absurdo, supongamos que L y $\overline{L}$ se hallan garantizados al mismo tiempo. Por simplicidad y sin pérdida de generalidad, supongamos además que L es de la forma $C(a)$. Entonces, bajo las hipótesis especificadas, tenemos que existen dos argumentos garantizados $\mathcal{A}$ y $\mathcal{B}$ tal que $\langle \mathcal{A}, C(a)\rangle$ y $\langle \mathcal{B}, \sim C(a)\rangle$. Sin embargo, como sus literales son complementarios, debe ser que $\langle \mathcal{B}, \sim C(a)\rangle$ es un derrotador para $\langle \mathcal{A}, C(a)\rangle$. Pero como $\langle \mathcal{A}, C(a)\rangle$ está garantizado, todos sus derrotadores se hallan derrotados. En particular, $\langle \mathcal{B}, \sim C(a)\rangle$ debe estar derrotado. Absurdo, pues $\langle \mathcal{B}, \sim C(a)\rangle$ está garantizado por hipótesis.* □

Propiedad 5.1.7 *Sea Σ una δ-ontología. No puede ser el caso que un individuo a pertenezca justificadamente a los conceptos C y $\neg C$ simultáneamente.*

Demostración: *Supongamos que $\Sigma = (T_S, T_D, A)$ y supongamos también que puede ocurrir C_j^a y $(\neg C)_j^a$ al mismo tiempo. Entonces debe ser el caso que existen dos argumentos garantizados $\langle \mathcal{A}, C(a)\rangle$ y $\langle \mathcal{B}, \sim C(a)\rangle$ con respecto al programa DeLP $(\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$. Pero esto es imposible ya que en DeLP no es posible garantizar dos literales complementarios simultáneamente (en virtud de la Propiedad 5.1.6).* □

Propiedad 5.1.8 *Sea Σ una δ-ontología. No puede ser el caso que un individuo a pertenezca estrictamente a los conceptos C y $\neg C$ simultáneamente.*

Demostración: *Supongamos que $\Sigma = (T_S, T_D, A)$ y supongamos también que puede ocurrir C_s^a y $(\neg C)_s^a$ al mismo tiempo. Entonces debe ser el caso que existen dos argumentos $\langle \emptyset, C(a)\rangle$ y $\langle \emptyset, \sim C(a)\rangle$ con respecto al programa DeLP $(\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$.*

Entonces debe ser el caso que $\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A)$ es inconsistente (contradiciendo la Definición 4.5.2). □

5.1.2. Consistencia de terminologías

Dada una δ-ontología $\Sigma = (T_S, T_D, A)$, en la Sección 4.5 propusimos definiciones para caracterizar las nociones de coherencia interna en la Abox A. En esta sección proponemos un conjunto de definiciones adicionales para caracterizar diferentes grados de consistencia en una δ-ontología y analizaremos algunas de las propiedades que se desprenden de tales consideraciones. Dichas propiedades tendrán profunda relación con la dinámica interna del razonamiento en la Programación en Lógica Rebatible. De esta manera, las propiedades halladas servirán para caracterizar a los sistemas de razonamiento con ontologías representadas en Lógicas para la Descripción utilizando formalismos de razonamiento no-monótonos.

Como veremos más adelante en este capítulo, es importante que un sistema no sea capaz de producir respuestas contradictorias (o no significativas) a partir de una ontología inconsistente. En esta sección damos un conjunto de definiciones para caracterizar formalmente diversos grados de consistencia a los que puede acatar una ontología y mostramos algunas de las propiedades que se desprenden de las mismas.

En la Sección 2.3.2 presentamos la noción de consistencia en el marco DL tradicional. Dicha noción dice que una Abox A es consistente con respecto a una Tbox T si y sólo si existe una interpretación que es modelo de A y T al mismo tiempo. En cambio, como en el acercamiento al razonamiento con ontologías inconsistentes propuesto en esta tesis, la Tbox T se haya particionada en una terminología estricta T_S y una terminología rebatible T_D y, además, debido a las restricciones impuestas en la representación de conocimiento en la Programación en Lógica Rebatible, en la Definición 4.5.1 exigimos que la Abox A fuera consistente con respecto a la terminología estricta; es decir, que a partir del programa DeLP que interpreta a $T_S \cup A$ no fuera posible derivar un par de literales complementarios. Tal noción de consistencia es extendida a continuación:

Definición 5.1.2 (Consistencia fuerte) *Sea $\Sigma = (T_S, T_D, A)$ una δ-ontología. Una ontología es fuertemente consistente si y sólo si la caja asercional A es consistente respecto de la terminología T_S. En caso contrario, diremos que la ontología es fuertemente inconsistente.*

Ejemplo 5.1.1 *La δ-ontología $\Sigma_{4.5.1}$ presentada en el Ejemplo 4.5.1 de la página 124 es fuertemente inconsistente ya que permite derivar estrictamente a los literales $F(a)$ y $\sim F(a)$ a partir del programa DeLP que la interpreta.*

En cambio la δ-ontología $\Sigma_{5.1.1} = (T_S, T_D, A)$*, donde:*

$$\begin{aligned} T_S &= \left\{ \begin{array}{l} D \sqsubseteq C \\ E \sqsubseteq D \end{array} \right\} \\ T_D &= \{\} \\ A &= \left\{ \; a : E \; \right\} \end{aligned}$$

es fuertemente consistente pues los únicos literales que son derivables en forma estricta a partir de $\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A)$ *son:* $\left\{ \; E(a), D(a), C(a) \; \right\}$.

Nótese que para evitar la formación de argumentos autoderrotados en DeLP, el formalismo impedirá la consideración de las δ-ontologías fuertemente inconsistentes. Esto es, una δ-ontología fuertemente inconsistente no será tratada, ya que el DeLP no considera válidos a los programas donde su parte estricta es inconsistente.

Definición 5.1.3 (Consistencia semi-fuerte) *Sea* $\Sigma = (T_S, T_D, A)$ *una δ-ontología. La δ-ontología* Σ *es semi-fuerte consistente si y sólo si no es posible determinar la pertenencia justificada de un individuo a a un par de conceptos complementarios C y* $\neg C$ *al mismo tiempo. En caso contrario, diremos que la ontología es semi-fuertemente inconsistente.*

Propiedad 5.1.9 *Todas las δ-ontologías son semi-fuerte consistentes.*

Demostración: *Sea* $\Sigma = (T_S, T_D, A)$ *una δ-ontología y sea* $\mathcal{P} = (\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$ *el programa DeLP interpretación de* Σ*. Sean C un nombre de concepto y a un nombre de individuo definidos en* Σ*. Supongamos por el absurdo que es posible determinar que* C_j^a *y* $(\neg C)_j^a$*. Si vale que* C_j^a*, entonces debe ser que existe un argumento garantizado* $\langle \mathcal{A}, C(a) \rangle$ *con respecto a* $\mathcal{P}$*. Entonces la respuesta las consultas* $C(a)$ *y* $\sim C(a)$ *serán* Sí *y* No*, respectivamente. Por otro lado, si vale que* $(\neg C)_j^a$*, entonces debe ser que existe un argumento garantizado* $\langle \mathcal{B}, \sim C(a) \rangle$ *con respecto a* $\mathcal{P}$*. Entonces la respuesta las consultas* $C(a)$ *y* $\sim C(a)$ *serán* No *y* Sí*, respectivamente. Esta situación es imposible pues DeLP no produce respuestas ambiguas.* □

Definición 5.1.4 (Consistencia débil) *Sea* $\Sigma = (T_S, T_D, A)$ *una δ-ontología. La δ-ontología* Σ *es débilmente consistente si y sólo si no es posible hallar un individuo a y un concepto C tal que sea posible determinar* C_p^a *y* $(\neg C)_p^a$ *al mismo tiempo. En caso contrario, diremos que la ontología es débilmente inconsistente.*

Ejemplo 5.1.2 *Sea la δ-ontología* $\Sigma_{5.1.2} = (\emptyset, T_D, A)$ *tal que:*

$$T_D = \left\{ \begin{array}{l} C \sqsubseteq D \\ D \sqsubseteq E \\ F \sqsubseteq \neg E \end{array} \right\}$$

$$A \;=\; \left\{ \begin{array}{l} a : C \\ a : F \end{array} \right\}$$

La interpretación de $\Sigma_{5.1.2}$ *es el programa DeLP* $\mathcal{P}_{5.1.2} = (\Pi, \Delta)$ *donde:*

$$\begin{array}{rcl} \Pi & = & \left\{ \; C(a), F(a) \; \right\} \\ \Delta & = & \left\{ \begin{array}{l} D(X) \prec C(X) \\ E(X) \prec D(X) \\ \sim E(X) \prec F(X) \end{array} \right\} \end{array}$$

A partir de $\mathcal{P}_{5.1.2}$ *es posible construir los argumentos* $\langle \mathcal{A}, E(a) \rangle$ *y* $\langle \mathcal{B}, \sim E(a) \rangle$ *donde:*

$$\begin{array}{rcl} \mathcal{A} & = & \left\{ \begin{array}{l} D(a) \prec C(a) \\ E(a) \prec D(a) \end{array} \right\} \; y \\ \mathcal{B} & = & \left\{ \; \sim E(a) \prec F(a) \; \right\}. \end{array}$$

Luego es posible determinar E_p^a *y* $(\neg E)_p^a$*. Por lo tanto,* $\Sigma_{5.1.2}$ *es débilmente inconsistente.*

Propiedad 5.1.10 *Si una ontología es débilmente consistente entonces es semi-fuerte consistente.*

Demostración: *Sea* $\Sigma = (T_S, T_D, A)$ *una δ-ontología. Sea* $\mathcal{P} = (\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$ *la interpretación de* Σ*. Si* Σ *es débilmente consistente entonces no existen ningún individuo* a *ni concepto* C *tal que vale que* C_p^a *y* $(\neg C)_p^a$*. Por lo tanto no existen dos argumentos* $\langle \mathcal{A}, C(a) \rangle$ *y* $\langle \mathcal{B}, \sim C(a) \rangle$*. Luego, tampoco es posible hallar dos argumentos garantizados* $\langle \mathcal{A}, C(a) \rangle$ *y* $\langle \mathcal{B}, \sim C(a) \rangle$*. Así, no es posible que se cumplan* C_j^a *y* $(\neg C)_j^a$ *al mismo tiempo. Luego,* Σ *es semi-fuerte consistente.* □

5.1.3. Reducibilidad de las inferencias

En las Lógicas para la Descripción tradicionales, dada una ontología $\Sigma = (T, A)$ se puede probar que un individuo a es instancia de un concepto C, notado como "$T \cup A \models a : C$", si y sólo si la Abox A se vuelve inconsistente al agregarle la aserción de concepto "$a : (\neg C)$", notado como "$T \cup A \cup \{a : (\neg C)\} \models \bot$", (Baader et al., 2003, Sección 2.2.4.3). En el marco de las δ-ontologías podemos hallar una propiedad similar:

Propiedad 5.1.11 *Sea* $\Sigma = (T_S, T_D, A)$ *una δ-ontología, sea* C *un nombre de concepto y* a *un nombre de individuo ambos definidos en* Σ*. Sea* $\Sigma' = (T_S, T_D, A \cup \{a : \neg C\})$*. Si vale que* C_s^a *con respecto a* Σ*, entonces* Σ' *es fuertemente inconsistente.*

Demostración: *Sea* $\mathcal{P} = (\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$*. Sea* $\mathcal{P}' = (\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A) \cup \mathfrak{T}_\Pi(\{a : \neg C\}), \mathfrak{T}_\Delta(T_D))$*. Si ocurre que* C_s^a *con respecto a* Σ*, entonces el literal* $C(a)$ *se deduce estrictamente a partir de* $\mathcal{P}$*. Como* $\mathfrak{T}_\Pi(\{a : \neg C\}) = \sim C(a)$*, el programa* $\mathcal{P}'$ *contiene al*

hecho $\sim C(a)$ *y, por contener al programa* $\mathcal{P}$*, contiene reglas que permiten derivar* $C(a)$*. Luego, debe ser que* Σ' *es fuertemente inconsistente.* □

Nótese que el dual de la Propiedad 5.1.11 (*i.e.*, que si Σ' es fuertemente inconsistente entonces C_s^a) no se cumple en el marco de las δ-ontologías mientras que sí lo es en las ontologías DL tradicionales. En efecto, en las DL tradicionales, si $A \cup \{a : (\neg C)\} \models \bot$, entonces como la base de conocimiento es inconsistente y las DL son isomorfas con la Lógica de Predicados reducida a dos variables es posible deducir cualquier sentencia a partir de tal ontología inconsistente. En particular, es posible deducir que $A \models a : C$. En cambio, en las δ-ontologías, como su semántica se basa en DeLP, si la Abox A es fuertemente inconsistente, no necesariamente se podrá deducir $a : C$ a partir de $A \cup \{a : \neg C\}$ como muestra el siguiente contraejemplo. Supongamos que $\Sigma = (T_S, T_D, A)$ es una δ-ontología, donde:

$$\begin{aligned} T_S &= \emptyset \\ T_D &= \emptyset \\ A &= \left\{ \begin{array}{l} b : C \\ b : \neg C \end{array} \right\} \end{aligned}$$

Si consideramos $\Sigma' = (T_S, T_D, A \cup \{a : \neg C\})$, ocurre que Σ' es fuertemente inconsistente, pues Σ ya lo era. En particular, ocurre que $(\neg C)_s^a$ con respecto a Σ' pero no ocurre que C_s^a con respecto a Σ'.

5.1.4. Proceso de razonamiento

Esta sección presentamos algunas propiedades adicionales concernientes al proceso de razonamiento en el marco de las δ-ontologías. Las propiedades presentadas están íntimamente ligadas a la dinámica del razonamiento en la Programación en Lógica Rebatible.

Propiedad 5.1.12 *Sea* $\Sigma = (T_S, T_D, A)$ *una* δ*-ontología. Si* A *es internamente incoherente entonces* $\mathfrak{T}(\Sigma)$ *no es un programa DeLP válido.*

Demostración: *Si la Abox* A *es internamente incoherente (veáse la Definición 4.5.1 en la página 124) entonces contiene dos aserciones de la forma* $a : C$ *y* $a : \neg C$*. Luego el programa DeLP* $\mathcal{P}$ *interpretación de* Σ*, con* $\mathcal{P} = (\Pi, \Delta)$ *donde* $\Pi = \mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A)$ *y* $\Delta = \mathfrak{T}_\Delta(T_D)$ *contiene dos hechos de la forma* $C(a)$ *y* $\sim C(a)$*. Por lo tanto, el conjunto* Π *es trivialmente inconsistente y* $\mathcal{P}$ *no es un programa DeLP válido.* □

Propiedad 5.1.13 *Sea* Σ *una* δ*-ontología donde la terminología estricta es vacía. Sea* $\mathcal{P} = \mathfrak{T}(\Sigma)$*. Si* Σ *es débilmente consistente, entonces los árboles dialécticos que se forman a partir de* $\mathcal{P}$ *tienen altura 0.*

Demostración: *Supongamos que* $\Sigma = (\emptyset, T_D, A)$. *Entonces* $\mathcal{P} = (\mathfrak{T}_\Pi(A), \mathfrak{T}_\Delta(T_D))$. *Sea* C *un nombre de concepto en* Σ. *Sea* a *un nombre de individuo en* Σ. *Pueden ocurrir una de dos situaciones:*

1. *No ocurre que* C_p^a: *En este caso no existe un argumento para* $C(a)$ *respecto del programa* $\mathcal{P}$.

2. *Ocurre que* C_p^a: *En este caso existe un argumento* $\langle \mathcal{A}, C(a) \rangle$ *respecto del programa* $\mathcal{P}$. *Sin embargo, no puede existir un derrotador directo para* $C(a)$ *ya que eso implicaría la existencia de un argumento* $\langle \mathcal{B}, \sim C(a) \rangle$. *Luego, sería el caso que* $(\neg C)_p^a$ *(contradiciendo la hipótesis que dice que* Σ *es débilmente consistente).*

   Tampoco puede existir un derrotador indirecto del argumento $\mathcal{A}$. *Supongamos que tal derrotador es* $\langle \mathcal{C}, D(b) \rangle$. *Luego, en la derivación de* $C(a)$, *existe un literal* $\sim D(b)$. *Así, debe existir un argumento* $\langle \mathcal{D}, \sim D(b) \rangle$. *Por lo tanto, ocurre que* D_p^b *y* $(\neg D)_p^b$ *(nuevamente contradiciendo la hipótesis que dice que* Σ *es débilmente consistente).*

Por lo tanto, para cada argumento que es posible construir a partir de $\mathcal{P}$, *no es posible hallar derrotadores. Por lo tanto, todos los argumentos que se pueden construir están garantizados y además tienen altura 0.* □

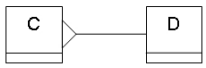
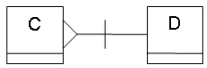
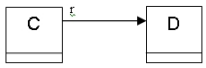
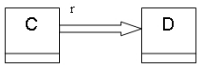
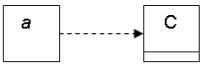
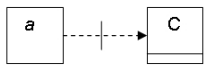
5.1.5. Patrones de razonamiento

En esta sección, exploramos patrones típicos de razonamiento que pueden surgir a la hora de representar conocimiento utilizando δ-ontologías. Los patrones considerados están inspirados en problemas bien conocidos. De manera de entender intuitivamente los patrones presentados, desarrollamos una notación gráfica que extiende la notación UML (por *Unified Modeling Language* o *Lenguaje de Modelado Unificado*) (Ambler, 2006; Miller, 2006; OMG, 2007). Como parte de las motivaciones para desarrollar una notación gráfica para las δ-ontologías, podemos decir que, en general, el ser humano, quizá por motivos evolutivos, es capaz de captar una gran cantidad de información a través de un diagrama mucho más fácilmente que, digamos, una especificación algebraica. Por otro lado, una motivación natural surge de que las Lógicas para la Descripción pueden ser consideradas una evolución natural de las Redes de Herencia (Horty et al., 1990). Por lo tanto, en todo caso, las δ-ontologías pueden, entonces, considerarse una evolución de las redes de herencia con excepciones (Al-Asady, 1993).

Sin embargo, también creemos que una notación gráfica debe estar apoyada en una semántica algebraica. Formalizamos entonces la respuesta de nuestro marco frente a tales patrones demostrando formalmente el origen de las mismas.

Notación gráfica de δ-ontologías: δ-UML

La notación que utilizaremos para describir tales patrones de δ-ontologías es una variación de la notación UML (Ambler, 2006; Miller, 2006; OMG, 2007); por este motivo, la denominamos *notación δ-UML* y la presentamos en la Tabla 5.1. Con motivo de esta discusión, el lenguaje subyacente utilizado para las δ-ontologías consideradas será un subconjunto de la Lógica para la Descripción $\mathcal{ALC}$ respetando las restricciones impuestas por la Definición 4.4.1.

Notación	*Lógicas para la Descripción*	*Significado*
	$(C \sqsubseteq D) \in T_S$	Subsunción de clases estricta
	$(C \sqsubseteq D) \in T_D$	Subsunción de clases rebatible
	$(C \sqsubseteq \neg D) \in T_S$	Exclusión de clases estricta
	$(C \sqsubseteq \neg D) \in T_D$	Exclusión de clases rebatible
	$(C \sqsubseteq \exists r.D) \in T_S$	Relación estricta
	$(C \sqsubseteq \exists r.D) \in T_D$	Relación tentativa
	$(a : C) \in A$	Instancia de concepto
	$(a : \neg C) \in A$	Instancia de complemento de concepto

Cuadro 5.1: Notación δ-UML para representar gráficamente δ-ontologías

Resumimos los elementos presentados en la notación δ-UML:

Subsunción de clases estricta: La subsunción estricta de un concepto C por otro concepto D implica que todos los individuos pertenecientes a la clase C pertenecen también a la clase D. En términos de δ-ontologías, esto quiere decir que el axioma $C \sqsubseteq D$ pertenece a la Sbox de la ontología en cuestión.

Subsunción de clases rebatible: La subsunción rebatible de un concepto C por otro concepto D implica que tentativamente los individuos pertenecientes a la clase C pertenecen también a la clase D (es decir, pueden existir excepciones a la relación de subsunción de clases). En términos de δ-ontologías, esto quiere decir que el axioma $C \sqsubseteq D$ pertenece a la Dbox de la ontología en cuestión.

Exclusión de clases estricta: En la exclusión de clases estricta entre los conceptos C y D entendemos que ningún individuo de C está incluido en el concepto D. En términos prácticos, esto quiere decir que el axioma $C \sqsubseteq \neg D$ está incluido en la Sbox de la ontología en cuestión.

Exclusión de clases rebatible: En la exclusión de clases rebatible entre los conceptos C y D entendemos que usualmente ningún individuo de C está incluido en el concepto D (es decir, la regla puede tener excepciones). En términos prácticos, esto quiere decir que el axioma $C \sqsubseteq \neg D$ está incluido en la Dbox de la ontología en cuestión.

Relación estricta: Por relación estricta por medio de un rol r entre las clases C y D, entendemos que siempre un individuo de clase C está relacionado por r con al menos un individuo de clase D. Esto quiere decir que el axioma $C \sqsubseteq \exists r.D$ está incluido en la Sbox de la ontología en cuestión.

Relación tentativa: Por relación tentativa por medio de un rol r entre las clases C y D, entendemos que usualmente un individuo de clase C está relacionado por r con al menos un individuo de clase D. Esto quiere decir que el axioma $C \sqsubseteq \exists r.D$ está incluido en la Dbox de la ontología en cuestión.

Instancia de concepto: Un individuo a es una instancia del concepto C si la aserción $a : C$ pertenece a la Abox de la ontología en cuestión.

Instancia de complemento de concepto: Un individuo a es una instancia del complemento del concepto C si la aserción $a : \neg C$ pertenece a la Abox de la ontología en cuestión.

Un patrón sistemáticamente nombra, motiva y explica un diseño general que trata un problema recurrente; describe el problema, su solución, cuando aplicar su solución y sus consecuencias. De esta manera, los patrones son medios eficientes de comunicación entre ingenieros de conocimiento; los patrones representan buena práctica, soluciones probadas y lecciones aprendidas (Tešanović, 2001). A continuación, estudiamos algunos de los patrones de razonamiento que pueden surgir al considerar la estructura sintáctica de una δ-ontología.

Primer patrón de razonamiento: Pingüinos voladores

Consideremos la siguiente δ-ontología $\Sigma_{penguin} = (\emptyset, T_D, A)$, donde:

$$\begin{aligned} T_D &= \left\{ \begin{array}{l} Bird \sqsubseteq Flies \\ Penguin \sqsubseteq Bird \\ Penguin \sqsubseteq \neg Flies \end{array} \right\} \\ A &= \left\{ \; opus : Penguin \; \right\} \end{aligned}$$

Cuando nos abstraemos de la naturaleza de los identificadores presentados, este patrón se puede representar como la δ-ontología $\Sigma_{pattern\text{-}1} = (\emptyset, T_D, A)$ que se presenta a continuación:

$$\begin{array}{rcl} T_D & = & \left\{ \begin{array}{l} D \sqsubseteq C \\ E \sqsubseteq D \\ E \sqsubseteq \neg C \end{array} \right\} \\ A & = & \left\{ \; x_1 : E \; \right\} \end{array}$$

En la Figura 5.1.(a), se puede apreciar la situación en forma gráfica. Enunciamos la siguiente propiedad que caracteriza el comportamiento de nuestro marco en presencia de tal patrón.

Propiedad 5.1.14 *Dado el patrón presentado en la Figura 5.1.(a), no es posible determinar que el individuo x_1 pertenece en forma justificada al concepto C.*

Demostración: *Bajo la suposición de que los identificadores considerados no aparecen en ninguna otra parte de la ontología, al considerar la interpretación de la δ-ontología que representa el patrón de la Figura 5.1.(a), se obtiene el siguiente programa lógico rebatible $\mathcal{P}_{pattern\text{-}1} = (\Pi, \Delta)$ tal que:*

$$\begin{array}{rcl} \Pi & = & \left\{ \; E(x_1) \; \right\} \\ \Delta & = & \left\{ \begin{array}{l} C(X) \prec D(X) \\ D(X) \prec E(X) \\ {\sim}C(X) \prec E(X) \end{array} \right\} \end{array}$$

Bajo la suposición de que los símbolos involucrados en $\mathcal{P}_{pattern\text{-}1}$ no aparecen en el resto de la ontología, es posible abstraerse del resto de la misma de tal manera que, a partir de tal programa, es posible construir dos argumentos $\langle \mathcal{A}_1, C(x_1) \rangle$ y otro $\langle \mathcal{A}_2, {\sim}C(x_1) \rangle$, con:

$$\begin{array}{rcl} \mathcal{A}_1 & = & \left\{ \begin{array}{l} C(x_1) \prec D(x_1) \\ D(x_1) \prec E(x_1) \end{array} \right\}, \; y \\ \mathcal{A}_2 & = & \left\{ \; {\sim}C(x_1) \prec E(x_1) \; \right\}. \end{array}$$

Estos dos argumentos se derrotan mutuamente adoptando los árboles de dialéctica involucrados la configuración siguiente:

$$\begin{array}{cc} \langle \mathcal{A}_1, C(x_1) \rangle & \langle \mathcal{A}_2, {\sim}C(x_1) \rangle \\ \Big| & \Big| \\ \langle \mathcal{A}_2, {\sim}C(x_1) \rangle & \langle \mathcal{A}_1, C(x_1) \rangle \end{array}$$

Por lo tanto, las respuestas a la consultas $C(x_1)$ y ${\sim}C(x_1)$ serán INDECISO *y no será posible determinar que x_1 pertenece en forma justificada al concepto C ni al concepto $\neg C$.* $\square$

Segundo patrón de razonamiento: Pingüinos voladores revisados

Al modificar levemente la ontología $\Sigma_{penguin}$ de tal manera de obtener la ontología $\Sigma_{penguin\text{-}rev} = (T_S, T_D, A)$ donde:

$$\begin{array}{rcl} T_S & = & \left\{ \; Penguin \sqsubseteq Bird \; \right\} \\ T_D & = & \left\{ \begin{array}{l} Bird \sqsubseteq Flies \\ Penguin \sqsubseteq \neg Flies \end{array} \right\} \\ A & = & \left\{ \; opus : Penguin \; \right\} \end{array}$$

Esta ontología puede ser abstraída por el siguiente patrón, que se presenta en forma gráfica en la Figura 5.1.(b) y que damos en la forma de la δ-ontología $\Sigma_{pattern\text{-}2} = (T_S, T_D, A)$ con:

$$\begin{array}{rcl} T_S & = & \left\{ \; E \sqsubseteq D \; \right\} \\ T_D & = & \left\{ \begin{array}{l} D \sqsubseteq C \\ E \sqsubseteq \neg C \end{array} \right\} \\ A & = & \left\{ \; x_1 : E \; \right\} \end{array}$$

Propiedad 5.1.15 *Dado el patrón presentado en la Figura 5.1.(b), el individuo x_1 pertenece en forma justificada al concepto $\neg C$.*

Demostración: *Bajo la suposición de que los identificadores considerados no aparecen en ninguna otra parte de la ontología, al considerar la interpretación de la δ-ontología $\Sigma_{pattern\text{-}2}$ que representa el patrón de la Figura 5.1.(b), se obtiene el siguiente programa lógico rebatible $\mathcal{P}_{pattern\text{-}2} = (\Pi, \Delta)$ tal que:*

$$\begin{array}{rcl} \Pi & = & \left\{ \begin{array}{l} D(X) \leftarrow E(X) \\ E(x_1) \end{array} \right\} \\ \Delta & = & \left\{ \begin{array}{l} C(X) \prec D(X) \\ \sim C(X) \prec E(X) \end{array} \right\} \end{array}$$

Con respecto a la consulta $C(x_1)$, se pueden construir dos argumentos a partir del programa previo $\mathcal{P}_{pattern\text{-}2}$, a saber, $\langle \mathcal{B}_1, C(x_1) \rangle$ y $\langle \mathcal{B}_2, \sim C(x_1) \rangle$, donde:

$$\begin{array}{rcl} \mathcal{B}_1 & = & \left\{ \; C(x_1) \prec D(x_1) \; \right\}, y \\ \mathcal{B}_2 & = & \left\{ \; \sim C(x_1) \prec E(x_1) \; \right\} \end{array}$$

En este caso el argumento $\mathcal{B}_2$ derrota a $\mathcal{B}_1$, dando lugar a los siguientes árboles de dialéctica:

$$\begin{array}{c} \langle \mathcal{B}_1, C(x_1) \rangle \\ \Big| \\ \langle \mathcal{B}_2, \sim C(x_1) \rangle \quad \langle \mathcal{B}_2, \sim C(x_1) \rangle \end{array}$$

Por lo tanto, como el argumento $\langle \mathcal{B}_2, \sim C(x_1) \rangle$ se halla garantizado, la respuesta a la consulta $C(x_1)$ planteada con respecto al programa $\mathcal{P}_{pattern\text{-}2}$ es NO. *Luego, el individuo x_1 pertenece en forma justificada al concepto $\neg C$.* $\square$

Tercer patrón de razonamiento: Aves anormales

Consideremos la siguiente ontología $\Sigma_{anormal} = (\emptyset, T_D, A)$ que expresa que las aves anormales no son capaces de volar mientras que las aves de las que no conocemos otra información asumimos que son capaces de volar. Además, Opus es un ave y también anormal.

$$\begin{array}{rcl} T_D & = & \left\{ \begin{array}{l} Bird \sqsubseteq Flies \\ Bird \sqcap Abnormal \sqsubseteq \neg Flies \end{array} \right\} \\ A & = & \left\{ \begin{array}{l} opus : Bird \\ opus : Abnormal \end{array} \right\} \end{array}$$

A partir de esta ontología, obtenemos el siguiente patrón (véase la Figura 5.1.(c)) que expresamos como la δ-ontología $\Sigma_{pattern\text{-}3} = (\emptyset, T_D, A)$, con:

$$\begin{array}{rcl} T_D & = & \left\{ \begin{array}{l} C \sqsubseteq D \\ C \sqcap E \sqsubseteq \neg D \end{array} \right\} \\ A & = & \left\{ \begin{array}{l} x_1 : C \\ x_1 : E \end{array} \right\} \end{array}$$

Propiedad 5.1.16 *Dado el patrón presentado en la Figura 5.1.(c), x_1 es un miembro justificado del concepto $\neg D$.*

Demostración: *La interpretación de la δ-ontología $\Sigma_{anormal}$ es el programa DeLP $\mathcal{P}_{anormal} = (\Pi, \Delta)$ donde:*

$$\begin{array}{rcl} \Pi & = & \left\{ \begin{array}{l} C(x_1) \\ E(x_1) \end{array} \right\}, y \\ \Delta & = & \left\{ \begin{array}{l} D(X) \prec C(X) \\ \sim D(X) \prec C(X), E(X) \end{array} \right\} \end{array}$$

Para determinar si si x_1 es un miembro justificado del concepto D, o que x_1 es un miembro justificado del concepto $\neg D$, construimos dos argumentos $\langle \mathcal{C}_1, D(x_1) \rangle$ y $\langle \mathcal{C}_2, \sim D(x_1) \rangle$, donde:

$$\begin{array}{rcl} \mathcal{C}_1 & = & \left\{ \begin{array}{l} D(x_1) \prec C(x_1) \end{array} \right\} \\ \mathcal{C}_2 & = & \left\{ \begin{array}{l} \sim D(x_1) \prec C(x_1), E(x_1) \end{array} \right\} \end{array}$$

En particular, el argumento $\mathcal{C}_2$ derrota a $\mathcal{C}_1$ dando lugar a los árboles de dialéctica siguientes:

$$\begin{array}{c} \langle \mathcal{C}_1, D(x_1) \rangle \\ \Big| \\ \langle \mathcal{C}_2, \sim D(x_1) \rangle \end{array} \qquad \langle \mathcal{C}_2, \sim D(x_1) \rangle$$

Así, el argumento $\langle \mathcal{C}_2, \sim D(x_1) \rangle$ se halla garantizado y el individo x_1 pertenece justificadamente al concepto $\neg D$. $\square$

Cuarto patrón de razonamiento: Diamante de Nixon

En cuarto lugar, consideremos la δ-ontología $\Sigma_{nixon\text{-}diamond} = (\emptyset, T_D, A)$ con:

$$\begin{array}{rcl} T_D & = & \left\{ \begin{array}{l} Quaker \sqsubseteq Pacifist \\ Republican \sqsubseteq \neg Pacifist \end{array} \right\} \\ A & = & \left\{ \begin{array}{l} nixon : Quaker \\ nixon : Republican \end{array} \right\} \end{array}$$

El patrón de razonamiento asociado a esta ontología es el presentado en la Figura 5.1.(d), y que a continuación expresamos como la δ-ontología $\Sigma_{pattern\text{-}4} = (\emptyset, T_D, A)$ con:

$$\begin{array}{rcl} T_D & = & \left\{ \begin{array}{l} D \sqsubseteq C \\ E \sqsubseteq \neg C \end{array} \right\} \\ A & = & \left\{ \begin{array}{l} x_1 : D \\ x_1 : E \end{array} \right\} \end{array}$$

Propiedad 5.1.17 *Dado el patrón presentado en la Figura 5.1.(d), el individuo x_1 no es un miembro justificado del concepto C ni de $\neg C$.*

Demostración: *La interpretación de la δ-ontología $\Sigma_{pattern\text{-}4}$ es el programa DeLP $\mathcal{P}_{pattern\text{-}4} = (\Pi, \Delta)$ donde:*

$$\begin{array}{rcl} \Pi & = & \left\{ \begin{array}{l} D(x_1) \\ E(x_1) \end{array} \right\} \\ \Delta & = & \left\{ \begin{array}{l} C(X) \prec D(X) \\ \sim C(X) \prec E(X) \end{array} \right\} \end{array}$$

A partir del mismo, es posible construir dos argumentos $\langle \mathcal{D}_1, C(x_1) \rangle$ y $\langle \mathcal{D}_2, \sim C(x_1) \rangle$ donde:

$$\begin{array}{rcl} \mathcal{D}_1 & = & \left\{ \; C(x_1) \prec D(x_1) \; \right\} \\ \mathcal{D}_2 & = & \left\{ \; \sim C(x_1) \prec E(x_1) \; \right\}, \end{array}$$

los cuales dan lugar a los siguientes árboles dialécticos:

$$\begin{array}{cc} \langle \mathcal{D}_1, C(x_1) \rangle & \langle \mathcal{D}_2, \sim C(x_1) \rangle \\ \Big| & \Big| \\ \langle \mathcal{D}_2, \sim C(x_1) \rangle & \langle \mathcal{D}_1, C(x_1) \rangle \end{array}$$

indicando que ambos se derrotan mutuamente. Por lo tanto, la respuesta la consulta $C(x_1)$ es Indeciso *y x_1 no pertenece en forma justificada ni al concepto C y ni tampoco a $\neg C$.* □

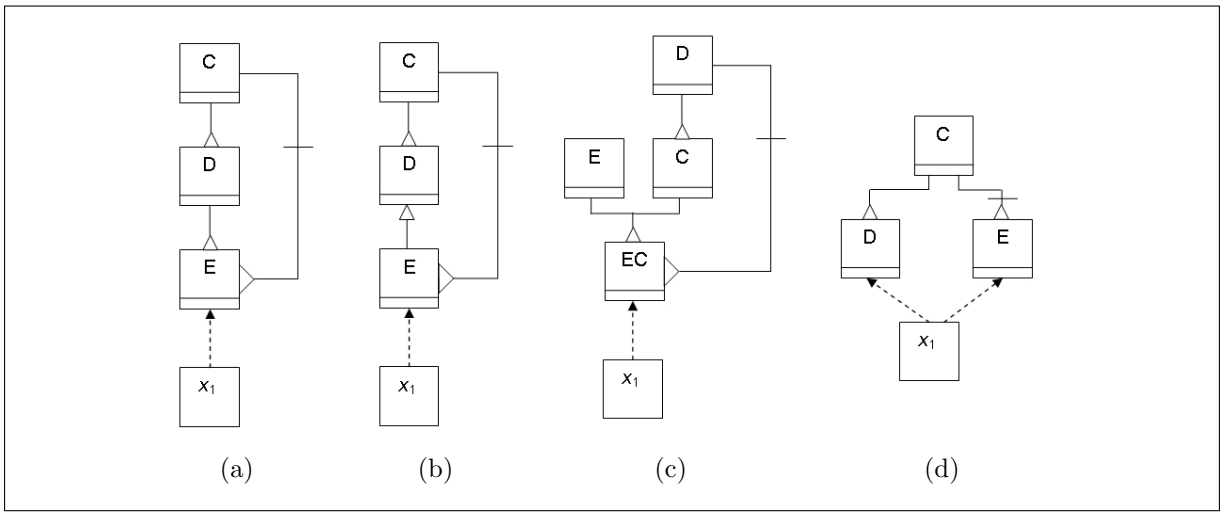


Figura 5.1: Patrones de razonamiento con δ-ontologías: (a) Primer patrón; (b) Segundo patrón; (c) Tercer patrón, y, (d) Cuarto patrón

5.1.6. Análisis de la complejidad con respecto al tiempo de ejecución

Uno de los objetivos de la iniciativa de la Web Semántica consiste en proveer métodos para deducir información a partir de ontologías que escalen al tamaño de la web. Cecchi *et al.* (Cecchi et al., 2006) han probado que los problemas de decisión de la determinación de si un conjunto de reglas rebatibles es un argumento para un literal bajo un programa DeLP es P-completo. Además, determinar si existe un argumento para un literal dado bajo un programa DeLP es NP.

Propiedad 5.1.18 *El tiempo de ejecución de los procesos para "recuperación abierta" y para "recuperación de todas las clases" es finito.*

Demostración: *Como una δ-ontología tiene un número finito de conceptos nombrados y de constantes de individuos, el programa DeLP obtenido a partir de la misma es finito. (Cecchi et al., 2006) han demostrado que determinar si existe un argumento para un literal es NP. Además, como el procedimiento de garantía siempre construye un árbol dialéctico finito, entonces los procesos "recuperación abierta" y "recuperación de todas las clases" siempre terminan.* □

Realizar un análisis exhaustivo de la complejidad temporal de la solución propuesta en esta tesis a la resolución de consultas en ontologías inconsistentes no es aún posible debido a que la complejidad temporal de los procesos dialécticos no se encuentra todavía entendida completamente. Parte de nuestro trabajo futuro está orientado a la resolución de estos problemas abiertos. En este sentido, es importante el aporte realizado por Krötzch et al. (2006), quienes analizan la complejidad de bases de conocimiento definidas

en Lógicas para la Descripción cuando las mismas son reexpresadas como programas en *Horn Description Logics*.

5.2. Propiedades respecto del razonamiento tradicional

5.2.1. Background: Un marco para la evaluación formal de la propuesta de integración

Como ya hemos comentado oportunamente, el marco de razonanamiento con δ-ontologías básicamente utiliza el lenguaje de las Lógicas para la Descripción para razonar con ontologías inconsistentes transformándolas en programas expresados en la Programación en Lógica Rebatible. Así, resulta interesante poder determinar cuál es la relación entre los dos formalismos de razonamiento (es decir, el de las Lógicas para la Descripción en la tradición de Baader et al. (2003) y el de las Lógicas para la Descripción embebidas en el marco de la Programación en Lógica Rebatible). Para ello, plantearemos un conjunto de propiedades deseables para caracterizar la relación entre el marco de razonamiento para ontologías DL propuesto en esta Tesis y aquel planteado por la escuela tradicional de razonamiento. Las definiciones que usaremos se basarán fuertemente en aquellas planteadas en los trabajos de Huang et al. (2004, 2005).

En particular, las especificaciones brindadas por (Huang et al., 2005) no se restringen a ningún lenguaje específico de representación de ontologías (aunque tienen a OWL en mente). En general, un lenguaje de ontologías puede ser considerado como un conjunto que puede ser generado a partir de reglas sintácticas. Así, consideran a una ontología como un conjunto de fórmulas. En su especificación se utiliza un operador de consecuencia no-clásico para razonar con inconsistencia, lo cual lo hace muy atractivo para la evaluación formal de nuestra propuesta de razonamiento con ontologías inconsistentes. En la siguiente discusión, usaremos $\models$ para denotar la relación de deducción clásica, y usaremos $\mid\!\approx$ para denotar una relación de deducción *no clásica*, la cual puede estar parametrizada para evitar ambigüedades, siguiendo la propuesta de Huang et al. (2004, 2005).

En razonamiento clásico, una consulta ϕ sobre una ontología Σ puede ser expresada como la evaluación de la relación de consecuencia $\Sigma \models \phi$. Hay solamente dos respuestas a dicha consulta: Si ($\Sigma \models \phi$) o No ($\Sigma \not\models \phi$). La respuesta Si significa que ϕ es una consecuencia lógica de Σ. La respuesta No significa que ϕ no puede ser deducido a partir de Σ. En particular la última respuesta no significa que la negación de ϕ valga en la ontología Σ, debido a que usualmente no se adopta la Hipótesis de Mundo Cerrado al utilizar una ontología (véase la Definición 3.1.1). Cuando se razone sobre una ontología Σ posiblemente inconsistente, Huang *et al.* distinguen cuatro estados epistémicos para las respuestas:

Definición 5.2.1 (Estado epistémico de una respuesta (Huang et al., 2004)) *Dada una ontología Σ y una consulta ϕ, la respuesta a la consulta ϕ tiene uno de los cuatro estados epistémicos:*

1. *Sobre-determinada:* $\Sigma \mid\!\approx \phi$ *y* $\Sigma \mid\!\approx \neg\phi$;
2. *Aceptada:* $\Sigma \mid\!\approx \phi$ *y* $\Sigma \not\mid\!\approx \neg\phi$;
3. *Rechazada:* $\Sigma \not\mid\!\approx \phi$ *y* $\Sigma \mid\!\approx \neg\phi$, *e*
4. *Indeterminada:* $\Sigma \not\mid\!\approx \phi$ *y* $\Sigma \not\mid\!\approx \neg\phi$.

Dado un razonador sobre inconsistencia, se espera que éste sea capaz de retornar respuestas significativas a las consultas en presencia de una ontología inconsistente. En el caso de una ontología consistente Σ, el razonamiento clásico es sensato, es decir una fórmula ϕ deducida a partir de Σ vale en cada modelo de Σ. Esta definición no es deseable para una ontología inconsistente Σ ya que toda fórmula se deduce a partir de ella al utilizar razonamiento clásico. Sin embargo, puede ocurrir que sólo una parte pequeña de Σ ha sido incorrectamente modelada o construida. Por lo tanto, Huang et al. (2005) proponen la siguiente definición de sensatez:

Definición 5.2.2 (Sensatez (*Soundness*) (Huang et al., 2005)) *Un razonador por inconsistencia $\mid\!\approx$ es sensato si las fórmulas que se deducen a partir de una teoría inconsistente Σ se pueden deducir a partir de una subteoría Σ' de Σ usando razonamiento clásico. Formalmente, vale la siguiente condición:*

$$\Sigma \mid\!\approx \phi \Rightarrow (\exists \Sigma' \subseteq \Sigma)(\Sigma' \not\models \bot \wedge \Sigma' \models \phi)$$

En otras palabras, las $\mid\!\approx$-consecuencias deben ser justificables en la base de un subconjunto consistente de la teoría. En particular, la definición previa no debería valer en la dirección opuesta. Si la implicación valiera en la dirección opuesta, tendríamos un razonador por inconsistencia que retornaría respuestas inconsistentes. Por ejemplo, si $\{a, \neg a\} \subseteq \Sigma$, entonces el razonador retornaría tanto a como $\neg a$ en presencia de Σ, lo cual representa una situación indeseable. Por lo tanto, el razonador por inconsistencia no debería retornar respuestas que se siguen a partir de *cualquier* subconjunto de Σ sino a partir de conjuntos *específicamente elegidos* de Σ.

Definición 5.2.3 (Consistencia (*Consistency*) (Huang et al., 2005)) *Un razonador por inconsistencia $\mid\!\approx$ es consistente si y sólo si* $\Sigma \mid\!\approx \phi \Rightarrow \Sigma \not\mid\!\approx \neg\phi$.

Definición 5.2.4 (Significatividad (*Meaningfulness*) (Huang et al., 2005)) *Una respuesta dada por un razonador por inconsistencia es significativa si y sólo si es consistente y sensata. Un razonador por inconsistencia se dice significativo si y sólo si todas sus respuestas son significativas.*

En virtud de la presencia de inconsistencias, la noción de completitud clásica es imposible. En consecuencia, (Huang et al., 2005) sugieren la noción de completitud local:

Definición 5.2.5 (Completitud local (*Local completeness*) (Huang et al., 2005)) *Un razonador por inconsistencia es localmente completo con respecto a una subteoría Σ' si y sólo si para cualquier fórmula ϕ se cumple la siguiente condición:*

$$\Sigma' \models \phi \Rightarrow \Sigma |\!\!\sim \phi.$$

Debido a que la condición anterior puede ser escrita como:

$$\Sigma \not|\!\!\sim \phi \Rightarrow \Sigma' \not\models \phi,$$

la completitud local puede considerarse como el complemento a la propiedad de sensatez.

Definición 5.2.6 (Maximalidad (*Maximality*) (Huang et al., 2005)) *Una razonador por inconsistencia es maximal si y sólo si existe una subteoría consistente maximal tal que su conjunto de consecuencias es el mismo que el conjunto de consecuencias del razonador por inconsistencias. Formalmente:*

$$\exists(\Sigma' \subseteq \Sigma)((\Sigma' \not\models \bot) \wedge (\forall \Sigma'' \supset \Sigma' \wedge \Sigma'' \subseteq \Sigma)(\Sigma'' \models \bot) \wedge \forall \phi(\Sigma' \models \phi \Leftrightarrow \Sigma |\!\!\sim \phi)).$$

Definición 5.2.7 (Consistencia local (*Local soundness*) (Huang et al., 2005)) *Una respuesta a una consulta $\Sigma |\!\!\sim \phi$ se dice localmente consistente con respecto a un conjunto $\Sigma' \subseteq \Sigma$, y si y sólo si se cumple la siguiente condición:*

$$\Sigma |\!\!\sim \phi \quad \Rightarrow \quad \Sigma' \models \phi.$$

Propiedad 5.2.1 (Huang et al., 2005)

(a) Sensatez local implica sensatez y significatividad.
(b) Completitud maximal implica completitud local.

5.2.2. Preservación de la semántica al transformar ontologías a DeLP

En esta sección estudiamos el análisis de la correspondencia entre las Lógicas para la Descripción y la Programación en Lógica Rebatible. Para ello usaremos el marco de evaluación de Huang et al. (2005) presentado en la sección previa. Como mencionamos oportunamente en el Capítulo 4, las funciones de transformación entre el lenguaje de las terminologías y las aserciones en Lógicas para la Descripción ya sea hacia reglas estrictas o rebatibles, están fundadas en investigaciones previas realizadas por (Grosof et al., 2003) y (Volz, 2004). En tales investigaciones, se propone una transformación entre las Lógicas

para la Descripción y la Programación en Lógica que preserva la semántica. En tal acercamiento se utiliza a la Lógica de Primer Orden como un mediador entre las dos lógicas. De la misma manera, nuestro acercamiento se funda en tales trabajos para transformar reglas de la Programación en Lógica en reglas de la Programación en Lógica Rebatible.

Al revisar la teoría subyacente a las Lógicas para la Descripción, en la Sección 2.3.3 (página 41), vimos que dada una expresión de clase C expresada en el lenguaje de las Lógicas para la Descripción es posible definir una transformación $\phi_C(x)$ para expresar C en la Lógica de Primer Orden utilizando una variable libre x. [1] La equivalencia entre ambas representaciones se define de la siguiente manera:

Definición 5.2.8 (Equivalencia entre clase DL y fórmula FOL (Borgida, 1996)) *Una clase C y su traducción $\phi_C(x)$ a una fórmula FOL son equivalentes si tenemos que para todas las interpretaciones $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ y para todo $a \in \Delta^{\mathcal{I}}$:*

$$a \in C^{\mathcal{I}} \text{ si } \mathcal{I} \models \phi_C(a)$$

Además, existe un resultado que relaciona ambas representaciones (Borgida, 1996) referido por (Volz, 2004, pág. 90).

Propiedad 5.2.2 ((Borgida, 1996)) *Una clase C es satisfacible si $\phi_C(x)$ es satisfacible.*

Con respecto a las transformaciones presentadas en la Sección 4.3, referimos al lector al trabajo de (Volz, 2004) donde se demuestra con detalle la validez de cada una de las transformaciones allí comentadas. Por una cuestión de claridad conceptual, presentamos una de las demostraciones allí mostradas.

Propiedad 5.2.3 ((Volz, 2004)) *La subsunción y la equivalencia de clases atómicas puede ser representada en el lenguaje de la programación en lógica.*

Demostración: *La prueba es una consecuencia inmediata de la traducción de las Lógicas para la Descripción en la Programación en Lógica:*

- *Axiomas de inclusión de clases:*

$$\phi(A \sqsubseteq B) = \forall x. \big(\ A(x) \rightarrow B(x) \ \big)$$

- *Axiomas de equivalencia de clases:*

$$\phi(A \equiv B) = \forall x. \big(\ (A(x) \rightarrow B(x)) \wedge (B(x) \rightarrow A(x)) \ \big)$$

- *Axiomas de inclusión de propiedades:*

[1] La aplicación recursiva de tal función puede generar la aparición de más variables libres.

$$\phi(P \sqsubseteq Q) = \forall x, y. \big(\ P(x, y) \rightarrow Q(x, y) \ \big)$$

- *Axiomas de equivalencia de propiedades:*

$$\phi(P \equiv Q) = \forall x, y. \big(\ (P(x, y) \rightarrow Q(x, y)) \wedge (Q(x, y) \rightarrow P(x, y)) \ \big)$$

□

Para interiorizarse sobre el resto de las demostraciones sobre la validez de las transformaciones de expresiones de conceptos expresadas en las Lógicas para la Descripción a expresiones expresadas en el lenguaje de la Programación en Lógica, referimos al lector a (Volz, 2004).

Para realizar las demostraciones correspondientes a la preservación de la semántica en las transformaciones de DL a DeLP, es necesario primero observar el siguiente lema:

Lema 5.2.1 *Sean C, C_1, C_2, D, D_1 y D_2 expresiones de conceptos.*

- *La expresión $C \equiv D$ es equivalente a dos expresiones de la forma $C \sqsubseteq D$ y $D \sqsubseteq C$.*
- *La expresión $C_1 \sqcup C_2 \sqsubseteq D$ es equivalente a dos expresiones de la forma $C_1 \sqsubseteq D$ y $C_2 \sqsubseteq D$.*
- *La expresión $C \sqsubseteq D_1 \sqcap D_2$ es equivalente a dos expresiones de la forma $C \sqsubseteq D_1$ y $C \sqsubseteq D_2$.*

Demostración: *Para realizar la demostración apelaremos a la relación entre las Lógicas para la Descripción (DL) y la Lógica de Predicados de Primer Orden (FOL) (véase la Sección 2.3.3 en la página 41). Demostramos cada uno de los casos:*

- *Un axioma DL de la forma $C \equiv D$ es una abreviatura para la sentencia FOL $(\forall x)(C(x) \leftrightarrow D(x))$. A su vez esta última es equivalente a $(\forall x)((C(x) \rightarrow D(x)) \wedge (D(x) \rightarrow C(x)))$, la que se puede escribir como dos sentencias $(\forall x)(C(x) \rightarrow D(x))$ y $(\forall x)(D(x) \rightarrow C(x))$. Estas dos últimas sentencias FOL se pueden expresar en el lenguaje de las DL como $C \sqsubseteq D$ y $D \sqsubseteq C$.*
- *Un axioma DL de la forma $C_1 \sqcup C_2 \sqsubseteq D$ es una abreviatura para la sentencia FOL $(\forall x)(((C_1(x) \vee C_2(x)) \rightarrow D(x))$. Ésta, por medio de las transformaciones de Lloyd-Topor, puede ser expresado como dos sentencias $(\forall x)((C_1(x) \rightarrow D(x))$ y $(\forall x)((C_2(x)) \rightarrow D(x))$, las que, a su vez, son equivalentes a los axiomas de inclusión DL $C_1 \sqsubseteq D$ y $C_2 \sqsubseteq D$.*
- *Un axioma DL de la forma $C \sqsubseteq D_1 \sqcap D_2$ es una abreviatura para la sentencia FOL $(\forall x)(C(x) \rightarrow (D_1(x) \wedge D_2(x)))$. Por medio de las transformaciones de Lloyd-Topor, ésta última puede ser expresada como dos sentencias $(\forall x)(C(x) \rightarrow D_1(x))$ y*

$(\forall x)(C(x) \rightarrow D_2(x))$, *las que, a su vez, son equivalentes a los axiomas de inclusión DL* $C \sqsubseteq D_1$ *y* $C \sqsubseteq D_2$.

□

Una consecuencia del trabajo de Volz (2004) es la siguiente propiedad:

Propiedad 5.2.4 *En presencia de ontologías conteniendo únicamente información positiva, las transformaciones* $\mathfrak{T}^*_{\Pi}$ *y* $\mathfrak{T}_{\Delta}$ *preservan la semántica de las terminologías DL transformadas.*

A continuación, mostramos solamente un *sketch* de la prueba.
Demostración: *Por inducción estructural en el sublenguaje de las Lógicas para la Descripción que se puede expresar en el lenguaje de la Programación en Lógica.*

- ***Axiomas de inclusión de clases:*** *Se puede mostrar, sin pérdida de generalidad (en virtud del Lema 5.2.1), que toda caja terminológica se puede considerar expresada como un conjunto de axiomas de la forma:*

$$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D$$

Así, cada axioma DL ψ *se convierte en uno o varias cláusulas de Horn de la forma* $D(X) \leftarrow C_1(X), \ldots, C_n(X)$ *que respetan la semántica de cada uno de los axiomas involucrados. Esto es así debido a la siguiente observación hecha en (Volz, 2004, Corolario 4.5.5) que dice que la aparición de* $\sqcap$ *en el miembro izquierdo de un axioma de inclusión DL es equivalente a una cierta fórmula en FOL. Formalmente:*

$$\phi(C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D) \;=\; \forall x.\left(\; C_1(x) \wedge \ldots \wedge C_n(x) \rightarrow D(x) \;\right).$$

Esta fórmula FOL es expresada en el caso de $\mathfrak{T}^*_{\Pi}$ *como:*

$$\mathfrak{T}^*_{\Pi}(C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D) \;=\; D(X) \leftarrow C_1(X), \ldots, C_n(X),$$

mientras que en el caso de $\mathfrak{T}_{\Delta}$ *es expresada como:*

$$\mathfrak{T}_{\Delta}(C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D) \;=\; D(X) \prec C_1(X), \ldots, C_n(X).$$

- ***Axiomas de inclusión y equivalencia de propiedades:*** *Los axiomas de inclusión de propiedades son interpretados de la siguiente manera:*

$$\phi(P \sqsubseteq Q) \;=\; (\forall x, y).\left(\; P(x,y) \rightarrow Q(x,y) \;\right)$$

los que son interpretados en DeLP como $Q(X,Y) \leftarrow P(X,Y)$ *o* $Q(X,Y) \prec P(X,Y)$.

Por otro lado, los axiomas de equivalencia de propiedades $P \equiv Q$ *son interpretados como dos axiomas de inclusión de propiedades* $P \sqsubseteq Q$ *y* $Q \sqsubseteq P$.

- ***Enumeraciones en el miembro izquierdo de axiomas de inclusión:*** *Las enumeraciones en el miembro izquierdo de un axioma de inclusión de clases son interpretadas de la siguiente manera:*

$$\phi(\{a_1, \ldots, a_n\} \sqsubseteq C) \;\; = \;\; (\forall x) \left(\; (x = a_1 \vee \ldots \vee x = a_1) \rightarrow C(x) \; \right)$$

las que en DeLP son interpretadas como n *reglas de la forma:* $C(X) \leftarrow X = a_i$ *con* $i = 1, \ldots, n$ *en las Sboxes y como* $C(X) \prec X = a_i$ *con* $i = 1, \ldots, n$ *en las Dboxes.*

- ***Transitividad de propiedades:*** *Los axiomas de transitividad de propiedades son interpretados de la siguiente manera:*

$$\phi(P^+ \sqsubseteq P) \;\; = \;\; (\forall x, y)(\exists z) \left(\; (P(x,z) \wedge P(z,y)) \rightarrow P(x,y) \; \right),$$

la que es interpretada con las reglas DeLP:

$$P(X,Y) \leftarrow P(X,Z), P(Z,Y) \text{ y}$$
$$P(X,Y) \prec P(X,Z), P(Z,Y).$$

- ***Aserciones de clases y propiedades:*** *Nótese que para las aserciones de la Abox, la preservación de la semántica se cumple trivialmente ya que* $a : C$ *y* $\langle a, b \rangle : R$ *son traducidos como* $C(a)$ *y* $R(a,b)$ *respectivamente.*

□

Propiedad 5.2.5 *La transformación* $\mathfrak{T}_\Pi$ *preserva la semántica.*

Demostración: *Dada una sentencia de inclusión de clases* $C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D$, *la transformación* $\mathfrak{T}_\Pi$ *se define como:*

$$\mathfrak{T}_\Pi(C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D) \;\; = \;\; \mathfrak{Transp}(\mathfrak{T}^*_\Pi(C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D))$$

Básicamente, esta transformación mantiene la semántica pues se basa en la observación siguiente. El axioma de inclusión DL:

$$C_1 \sqcap \ldots \sqcap C_i \sqcap \ldots \sqcap C_n \sqsubseteq D$$

es equivalente al axioma de inclusión DL:

$$C_1 \sqcap \ldots \sqcap \neg D \sqcap \ldots \sqcap C_n \sqsubseteq \neg C_i,$$

para todo $i = 1, \ldots, n$. *A partir de la regla estricta:*

$$D(X) \leftarrow C_1(X), \ldots, C_i(X), \ldots, C_n(X)$$

se generan todas las reglas:

$$\sim C_i(X) \leftarrow C_1(X), \ldots, \sim D(X), \ldots, C_n(X)$$

para todo $i = 1, \ldots, n$. □

5.2.3. Evaluación del marco de razonamiento con δ-ontologías

En vista del marco de evaluación propuesto por Huang et al. (2005), presentado en la Sección 5.2.1, evaluaremos entonces nuestro acercamiento al razonamiento con ontologías DL inconsistentes en términos del uso de δ-ontologías.

El trabajo de Huang básicamente se refiere a la relación existente entre dos formalismos en términos de las conclusiones que se pueden obtener respecto de la pertenencia de individuos a conceptos. Por un lado, se considera una relación de inferencia "tradicional", notada como $\models$. Esta relación sólo es capaz de obtener consecuencias en presencia de ontologías "consistentes". Por otro lado, se considera una una relación de inferencia "no estándar", notada como $\mid\!\approx$. Esta relación es capaz de obtener consecuencias "significativas" en presencia de ontologías "inconsistentes".

En nuestro contexto de trabajo, la relación $\models$ de Huang et al. (2005) se referirá a la relación de *chequeo de instancia* en (Baader et al., 2003). En cambio, la situación respecto de la relación $\mid\!\approx$ requiere de una adaptación ya que nuestro acercamiento con δ-ontologías brinda tres relaciones de inferencia en lugar de una: (i) pertenencia potencial de individuos a conceptos; (ii) pertenencia justificada de individuos a conceptos, y, (iii) pertenencia estricta de individuos a conceptos. Sin embargo y a pesar de que exploraremos las tres relaciones de inferencia en el marco propuesto por (Huang et al., 2005), cuando pensemos en las restricciones impuestas para la relación $\mid\!\approx$, lo haremos pensando en la relación de pertenencia justificada de individuos a conceptos (*i.e.*, la especificada en el inciso (ii)).

A continuación definimos formalmente la relación de inferencia en Lógicas para la Descripción:

Definición 5.2.9 (Relación de inferencia en DL) *Sea* Σ *una ontología DL,* C *un nombre de concepto en* Σ *y* a *un nombre de individuo en* Σ. *La relación de inferencia en Lógicas para la Descripción "*$\models$*" se define como:* $\Sigma \models C(a)$ *si y sólo si el individuo* a *es una instancia del concepto* C.

Seguidamente, definimos las relaciones de inferencia para δ-ontologías:

Definición 5.2.10 (Relación de pertenencia estricta en δ-ontologías) *Sea $\Sigma = (T_S, T_D, A)$ una δ-ontología, C un nombre de concepto en Σ, y a un nombre de individuo en Σ. La relación de pertenencia estricta del individuo a al concepto C con respecto a la ontología Σ,* i.e. *C_s^a, se denota como $\Sigma \mid\!\!\approx_S C(a)$.*

Definición 5.2.11 (Relación de pertenencia justificada en δ-ontologías) *Sea $\Sigma = (T_S, T_D, A)$ una δ-ontología, C un nombre de concepto en Σ, y a un nombre de individuo en Σ. La relación de pertenencia justificada del individuo a al concepto C con respecto a la ontología Σ,* i.e. *C_j^a, se denota como $\Sigma \mid\!\!\approx_J C(a)$.*

Definición 5.2.12 (Relación de pertenencia potencial en δ-ontologías) *Sea $\Sigma = (T_S, T_D, A)$ una δ-ontología, C un nombre de concepto en Σ, y a un nombre de individuo en Σ. La relación de pertenencia potencial del individuo a al concepto C con respecto a la ontología Σ,* i.e. *C_p^a, se denota como $\Sigma \mid\!\!\approx_P C(a)$.*

En presencia de inconsistencias, las conclusiones en ambos formalismos se obtendrán a partir de subconjuntos de la información presente. Por ello, definimos las nociones de subontologías en ambos formalismos:

Definición 5.2.13 (Subontología) *Sea $\Sigma = (T, A)$ una ontología DL. $\Sigma' = (T', A')$ es una subontología de Σ si y sólo si $T' \subseteq T$ y $A' \subseteq A$.*

Definición 5.2.14 (δ-subontología) *Sea $\Sigma = (T_S, T_D, A)$ una δ-ontología. $\Sigma' = (T'_S, T'_D, A')$ es una δ-subontología de Σ si y sólo si $T'_S \subseteq T_S$, $T'_D \subseteq T_D$ y $A' \subseteq A$.*

La siguiente es una propiedad que se desprende de la definición previa:

Propiedad 5.2.6 *Si Σ es un δ-ontología internamente coherente entonces cualquier δ-subontología de Σ también lo es.*

Demostración: *Supongamos que $\Sigma = (T_S, T_D, A)$. Sea $\Sigma' = (T'_S, T'_D, A')$ una δ-subontología de Σ. Si Σ es internamente coherente, entonces $\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A) \not\vdash \bot$. Por lo tanto, $\mathfrak{T}_\Pi(T'_S) \cup \mathfrak{T}_\Pi(A') \not\vdash \bot$. Luego, Σ' es internamente coherente.* □

En esta sección consideramos algunas de las propiedades que se desprenden del marco de razonamiento con δ-ontologías cuando las evaluamos con el marco propuesto por Huang et al. (2005). En virtud de las Definiciones 4.5.4 y 5.2.11, si consideramos a la relación "$\mid\!\!\approx_J$" de pertenencia justificada de instancias a conceptos, entonces $\Sigma \mid\!\!\approx \phi$ corresponde a la respuesta SI para la consulta ϕ realizada con respecto al programa DeLP $\mathfrak{T}(\Sigma)$. La primera propiedad que podemos enumerar se halla relacionada a la observación que en DeLP no es posible garantizar dos literales complementarios (véase la Propiedad 5.1.6).

Definición 5.2.15 (Ontología DL asociada a una δ-ontología) *Sea $\Sigma_\delta = (T_S, T_D, A)$ una δ-ontología. La ontología DL asociada a Σ_δ se define como:*

$$\mathfrak{AsocDL}(\Sigma_\delta) \;=\; (T_S \cup T_D, A).$$

Definición 5.2.16 (δ-ontología rebatible asociada a una ontología DL) *Sea $\Sigma_{DL} = (T, A)$ una ontología DL. La δ-ontología rebatible asociada a $\Sigma_{DL} = (T, A)$ se define como:*

$$\mathfrak{AsocDefeasible}(\Sigma_{DL}) \;=\; (\emptyset, T, A).$$

Definición 5.2.17 (δ-ontología estricta asociada a una ontología DL) *Sea $\Sigma_{DL} = (T, A)$ una ontología DL. La δ-ontología estricta asociada a $\Sigma_{DL} = (T, A)$ se define como:*

$$\mathfrak{AsocStrict}(\Sigma_{DL}) \;=\; (T, \emptyset, A).$$

Definición 5.2.18 (δ-ontología asociada a una ontología DL por una función de partición) *Sea $\Sigma_{DL} = (T, A)$ una ontología DL. Sea $\mathfrak{Sep}$ una función de partición de una terminología en una Sbox y una Dbox. Si $\mathfrak{Sep}(T) = (T_S, T_D)$, entonces la δ-ontología estricta asociada a $\Sigma_{DL} = (T, A)$ se define como:*

$$\mathfrak{AsocPart}_{\mathfrak{Sep}}(\Sigma_{DL}) \;=\; (T_S, T_D, A).$$

Las siguientes definiciones caracterizan el proceso de deducción en ontologías en Lógicas para la Descripción tradicionales.

Definición 5.2.19 (Deducción en una ontología DL) *Sea $\Sigma_{DL} = (T, A)$ una ontología DL. Sea C un nombre de concepto en Σ_{DL}. Sea a un nombre de individuo en Σ_{DL}. (a, C) se deduce de Σ_{DL} si y sólo si a es una instancia de C en Σ_{DL}. A la deducción en ontologías DL la notamos como $\Sigma_{DL} \models C(a)$.*

Definición 5.2.20 (Conjunto de respuestas de una ontología DL) *Sea $\Sigma_{DL} = (T, A)$ una ontología DL. Sea C un nombre de concepto en Σ_{DL}. Sea a un nombre de individuo en Σ_{DL}. El conjunto de respuestas de la ontología Σ_{DL} se define como:*

$$\mathfrak{DLAnswers}(\Sigma_{DL}) \;=\; \left\{\; C(a) | \Sigma_{DL} \models C(a) \;\right\}.$$

Las siguientes definiciones caracterizan el proceso de deducción en δ-ontologías.

Definición 5.2.21 (Conjunto de respuestas estrictas de una δ-ontología) *Sea $\Sigma_\delta = (T_S, T_D, A)$ una δ-ontología. Sea C un nombre de concepto en Σ_δ. Sea a un nombre de individuo en Σ_δ. El conjunto de respuestas estrictas de la ontología Σ_δ se define como:*

$$\mathfrak{StrictAnswers}(\Sigma_\delta) \;=\; \left\{\; C(a) | \Sigma_\delta \mid\!\!\approx_S C(a) \;\right\}.$$

Definición 5.2.22 (Conjunto de respuestas potenciales de una δ-ontología) *Sea $\Sigma_\delta = (T_S, T_D, A)$ una δ-ontología. Sea C un nombre de concepto en Σ_δ. Sea a un nombre de individuo en Σ_δ. El conjunto de respuestas potenciales de la ontología Σ_δ se define como:*

$$\mathfrak{PotentialAnswers}(\Sigma_\delta) \;=\; \left\{\; C(a) | \Sigma_\delta \mid\!\!\approx_P C(a) \;\right\}.$$

Definición 5.2.23 (Conjunto de respuestas justificadas de una δ-ontología) *Sea $\Sigma_\delta = (T_S, T_D, A)$ una δ-ontología. Sea C un nombre de concepto en Σ_δ. Sea a un nombre de individuo en Σ_δ. El conjunto de respuestas justificadas de la ontología Σ_δ se define como:*

$$\mathfrak{JustifiedAnswers}(\Sigma_\delta) \;=\; \left\{\; C(a) | \Sigma_\delta \mid\!\!\approx_J C(a) \;\right\}.$$

Propiedad 5.2.7 *Sea $\mid\!\!\approx_J$ la relación de pertenencia justificada de instancias a conceptos. Sea Σ una δ-ontología. La respuesta a una consulta ϕ nunca es sobre-determinada.*

Demostración: *Supongamos por el absurdo que la respuesta a la consulta ϕ está sobre-determinada. Entonces debe ser el caso que existen dos argumentos garantizados $\langle \mathcal{A}, \phi \rangle$ y $\langle \mathcal{A}, \sim\phi \rangle$ con respecto al programa DeLP $\mathfrak{T}(\Sigma)$. Contradicción: Esta situación no puede ocurrir pues contradice la Propiedad 5.1.6.* □

Note que, como es requerido por el razonamiento tradicional en las Lógicas para la Descripción, DeLP no adopta la hipótesis de mundo cerrado (véase la Definición 3.1.1 en la página 63). Esto es, no ser capaces de probar Q en DeLP no implica que $\sim Q$ será asumido. Por el contrario, tal respuesta será el resultado de un análisis dialéctico que tomará en cuenta todas las razones a favor de Q y de $\sim Q$ respectivamente.

Propiedad 5.2.8 *La relación de pertenencia justificada $\mid\!\!\approx_J$ es una relación de inferencia sensata (de acuerdo a la Definición 5.2.2).*

Demostración: *Sea $\Sigma = (T_S, T_D, A)$ una δ-ontología. Si $\Sigma \mid\!\!\approx_J C(a)$ entonces existe un argumento garantizado $\langle \mathcal{A}, C(a) \rangle$ con respecto a $\mathfrak{T}(\Sigma) = (\Pi_S \cup \Pi_A, \Delta)$, donde $\Pi_S = \mathfrak{T}_\Pi(T_S)$, $\Pi_A = \mathfrak{T}_\Pi(A)$ y $\Delta = \mathfrak{T}_\Delta(T_D)$. De acuerdo a la definición de argumento (véase la Definición 3.3.8), el conjunto $\Pi_S \cup \Pi_A \cup \mathcal{A}$ es consistente y como $\mathfrak{T}$ es una transformación que preserva la semántica (Grosof et al., 2003), debe existir un δ-subontología Σ' de Σ tal que $\mathfrak{T}(\Sigma') = \Pi_S \cup \Pi_A \cup \mathcal{A}$. Si consideramos a la ontología asociada DL $\Sigma_{DL} = \mathfrak{AsocDL}(\Sigma')$, a partir de ella se puede demostrar que a es una instancia del concepto C.* □

Propiedad 5.2.9 *La relación de pertenencia justificada $\mid\!\!\approx_J$ es una relación de inferencia consistente (en el sentido de la Definición 5.2.3).*

Demostración: *Sea Σ una δ-ontología. De acuerdo a la Propiedad 5.2.7, la relación de pertenencia justificada $\mid\!\!\approx_J$ nunca es sobre-determinada; por lo tanto, no puede ser el caso que $C(a) \in \mathfrak{JustifiedAnswers}(\Sigma)$ y que también $\neg C(a) \in \mathfrak{JustifiedAnswers}(\Sigma)$. Por lo tanto si $\Sigma \mid\!\!\approx_J C(a)$, entonces $\Sigma \not\mid\!\!\approx_J \neg C(a)$. Luego, la relación de pertenencia justificada $\mid\!\!\approx_J$ es una relación de inferencia consistente.* □

Propiedad 5.2.10 *La relación de pertenencia justificada* $\mid\!\!\approx_J$ *es una relación de inferencia significativa (en el sentido de la Definición 5.2.4).*

Demostración: *Este resultado se deduce a partir de las Propiedades 5.2.8 y 5.2.9.* □

Propiedad 5.2.11 *La relación de pertenencia potencial* $\mid\!\!\approx_P$ *es una relación de inferencia sensata (de acuerdo a la Definición 5.2.2).*

Demostración: *Dada una δ-ontología* Σ*, y dados un concepto* C *y un individuo* a*, debemos probar que si* C_p^a*, entonces existe una subontología* Σ' *de* $\mathfrak{AsocDL}(\Sigma)$ *tal que* $\Sigma' \models C(a)$*. Para ello, supongamos que* $\Sigma = (T_S, T_D, A)$*. Si* $\Sigma \mid\!\!\approx_P C(a)$ *entonces existe un argumento* $\langle \mathcal{A}, C(a)\rangle$ *con respecto a* $\mathfrak{T}(\Sigma) = (\Pi_S \cup \Pi_A, \Delta)$*, donde* $\Pi_S = \mathfrak{T}_\Pi(T_S)$*,* $\Pi_A = \mathfrak{T}_\Pi(A)$ *y* $\Delta = \mathfrak{T}_\Delta(T_D)$*. De acuerdo a la definición de argumento (véase la Definición 3.3.8), el conjunto* $\Pi_S \cup \Pi_A \cup \mathcal{A}$ *es consistente y como* $\mathfrak{T}$ *es una transformación que preserva la semántica (Grosof et al., 2003), debe existir un δ-subontología* Σ' *de* Σ *tal que* $\mathfrak{T}(\Sigma') = \Pi_S \cup \Pi_A \cup \mathcal{A}$*. Si consideramos a la ontología asociada DL* $\Sigma_{DL} = \mathfrak{AsocDL}(\Sigma')$*, a partir de ella se puede demostrar que* a *es una instancia del concepto* C*.* □

Note que la relación de pertenencia potencial no es en general una relación consistente (en el sentido de la Definición 5.2.3), como muestra el siguiente ejemplo:

Ejemplo 5.2.1 *Considere la siguiente δ-ontología* $\Sigma_{5.2.1} = (\emptyset, T_D, A)$ *donde* $T_D = \{(C \sqsubseteq D), (E \sqsubseteq \neg D)\}$ *y* $A = \{a : C, a : E\}$*. Es fácil ver que* D_p^a *y* $(\neg D)_p^a$*.*

Por lo tanto, la relación de pertenencia potencial tampoco es significativa (en el sentido de la Definición 5.2.4).

Propiedad 5.2.12 *La relación de pertenencia estricta* $\mid\!\!\approx_S$ *es una relación de inferencia sensata (de acuerdo a la Definición 5.2.2).*

Demostración: *Sea* $\Sigma = (T_S, T_D, A)$ *una δ-ontología. Si* $\Sigma \mid\!\!\approx_S C(a)$ *entonces existe un argumento* $\langle \emptyset, C(a)\rangle$ *con respecto a* $\mathfrak{T}(\Sigma) = (\Pi_S \cup \Pi_A, \Delta)$*, donde* $\Pi_S = \mathfrak{T}_\Pi(T_S)$*,* $\Pi_A = \mathfrak{T}_\Pi(A)$ *y* $\Delta = \mathfrak{T}_\Delta(T_D)$*. De acuerdo a la definición de argumento (véase la Definición 3.3.8), el conjunto* $\Pi_S \cup \Pi_A$ *es consistente y como* $\mathfrak{T}$ *es una transformación que preserva la semántica (Grosof et al., 2003), debe existir un δ-subontología* Σ' *de* Σ *tal que* $\mathfrak{T}(\Sigma') = \Pi_S \cup \Pi_A$*. Si consideramos a la ontología asociada DL* $\Sigma_{DL} = \mathfrak{AsocDL}(\Sigma')$*, a partir de ella se puede demostrar que* a *es una instancia del concepto* C*.* □

Propiedad 5.2.13 *La relación de pertenencia estricta* $\mid\!\!\approx_S$ *es una relación de inferencia consistente (de acuerdo a la Definición 5.2.3).*

Demostración: *Sea* Σ *una δ-ontología. Sea* $\mathcal{P} = \mathfrak{T}(\Sigma) = (\Pi, \Delta)$*. Sean* C *un nombre de concepto y* a *un nombre de individuo. Como* Π *es consistente, no puede ocurrir que a partir*

de Π *se deriven* $C(a)$ *y* $\sim C(a)$ *simultáneamente. Por lo tanto, si* $C(a) \in \mathfrak{StrictAnswers}(\Sigma)$, *entonces no puede ocurrir que* $\sim C(a) \in \mathfrak{StrictAnswers}(\Sigma)$. □

Propiedad 5.2.14 *La relación de pertenencia estricta* $\approx_S$ *es una relación de inferencia significativa (de acuerdo a la Definición 5.2.4).*

Demostración: *Este resultado se deduce a partir de las Propiedades 5.2.12 y 5.2.13.* □

La siguiente propiedad caracteriza la relación entre las deducciones posibles en las Lógicas para la Descripción tradicionales con respecto a las deducciones posibles en el marco de las δ-ontologías en presencia de δ-ontologías consistentes.

Propiedad 5.2.15 *Sea* $\Sigma_\delta = (T_S, T_D, A)$ *una* δ*-ontología débilmente consistente. Sea* $\Sigma_{DL} = \mathfrak{AsocDL}(\Sigma_\delta)$. *Entonces,* $\mathfrak{JustifiedAnswers}(\Sigma_\delta) \subseteq \mathfrak{DLAnswers}(\Sigma_{DL})$.

Demostración: *Si* Σ_δ *es débilmente consistente, entonces* Σ_{DL} *es consistente. Si* C *es un nombre de concepto y* a *es un individuo. Cuando* $C(a) \in \mathfrak{JustifiedAnswers}(\Sigma_\delta)$, *entonces existe un argumento garantizado* $\langle \mathcal{A}, C(a)\rangle$ *respecto del programa DeLP* $\mathfrak{T}(\Sigma_\delta)$. *Como* $\mathfrak{T}$ *es una transformación que preserva la semántica, debe ser que* $\mathfrak{T} \models C(a)$. *Luego* $C(a) \in \mathfrak{DLAnswers}(\Sigma_{DL})$. □

Ahora demostraremos la intuición que dice que en presencia de ontologías consistentes, todas las respuestas potenciales son respuestas justificadas. La demostración se basa en la observación de que, como la ontología es consistente, no es posible deducir pares de literales complementarios en su programa DeLP interpretación. Así, para cualquier argumento que se pueda formar a partir de tal programa, no es posible construir derrotadores. Por lo tanto, todo argumento está justificado. Formalmente:

Propiedad 5.2.16 *Sea* Σ_δ *una* δ*-ontología débilmente consistente. Entonces,*

$$\mathfrak{PotentialAnswers}(\Sigma_\delta) = \mathfrak{JustifiedAnswers}(\Sigma_\delta).$$

Demostración:

- ***Caso*** $\subseteq$***:*** *Supongamos que* $C(a) \in \mathfrak{PotentialAnswers}(\Sigma_\delta)$. *En particular, como* Σ_δ *es una ontología débilmente consistente, no puede ocurrir que* $\neg C(a) \in \mathfrak{PotentialAnswers}(\Sigma_\delta)$. *Además, por ser* Σ_δ *consistente, no es posible derivar literales complementarios a partir de* $\mathfrak{T}(\Sigma_\delta)$. *Luego, dado un argumento* $\langle \mathcal{A}, C(a)\rangle$ *obtenido a partir de* $\mathfrak{T}(\Sigma_\delta)$, *no es posible hallar un derrotador para el mismo. Por lo tanto,* $\langle \mathcal{A}, C(a)\rangle$ *se halla garantizado. Luego,* $C(a) \in \mathfrak{JustifiedAnswers}(\Sigma_\delta)$.

- ***Caso*** $\supseteq$***:*** *Trivialmente, una respuesta justificada es una respuesta potencial, ya que una respuesta justificada está soportada por un argumento justificado, el cual es un argumento y, por lo tanto, si* $C(a) \in \mathfrak{JustifiedAnswers}(\Sigma_\delta)$, *entonces* $C(a) \in \mathfrak{PotentialAnswers}(\Sigma_\delta)$.

□

Corolario 5.2.1 *Sea Σ una δ-ontología débilmente consistente. Entonces:*

$$\mathfrak{PotentialAnswers}(\Sigma) \subseteq \mathfrak{DLAnswers}(\mathfrak{AsocDL}(\Sigma)).$$

5.3. Discusión: Aplicaciones a la mezcla de ontologías

La *mezcla/integración* (en inglés, *merging/integration*) es el proceso de crear una nueva ontología a partir de dos o más ontologías existentes con partes solapadas (Klein, 2001). Adaptamos el marco de razonamiento con δ-ontologías presentado oportunamente en el Capítulo 4 para utilizarlo en la mezcla de ontologías.

Por lo tanto, en este contexto estamos interesados en modelar conocimiento ontológico con una δ-ontología. Luego, se recibirá una segunda ontología que propone una explicación para una definición alternativa de un concepto de la primera ontología. La aceptación de tal explicación puede producir una inconsistencia. Para ello, bajo una suposición de nombres únicos, interpretaremos ambas ontologías como programas DeLP. El conocimiento rebatible se une; sin embargo, el conocimiento estricto se unirá utilizando un *operador de mezcla* como el propuesto por las Teorías de Revisión de Creencias (Alchourron et al., 1985). La propuesta presentada en esta sección es una extensión de la presentada oportunamente en (Gómez et al., 2005*b*).

5.3.1. Marco de revisión de creencias

Los *sistemas de revisión de creencias* (Fermé, 2005) son marcos lógicos para modelar la dinámica del conocimiento. En diálogos entre dos agentes, es muy común que un agente no acepte completamente toda la información proporcionada por el otro, sino más bien partes de la misma. Un *operador de revisión* es una función que mapea conjuntos de sentencias K y A a un nuevo conjunto de sentencias. En particular, en (Falappa et al., 2002), el mecanismo de un operador de revisión $K \circ A$ por un conjunto de sentencias con aceptación parcial es definido de la siguiente manera: (1) el conjunto de entrada A es inicialmente aceptado, y (2) todas las posibles inconsistencias de $K \cup A$ son eliminadas. El mecanismo de este operador consiste de agregar A a K y entonces eliminar a partir del resultado todas las posibles inconsistencias por medio de una función de incisicon que hace un "corte" sobre cada conjunto minimalmente inconsistente de $K \cup A$ (Falappa et al., 2002).

En Falappa et al. (2002), las creencias son particionadas en dos conjuntos distinguidos: (1) *creencias particulars* K_P, que son representadas por hechos fijos, y (2) *creencias generales* K_G, que son representadas por implicaciones materiales cerradas. Así, cada base de creencias K tiene la forma $K_P \cup K_G$ donde $K_P \cap K_G = \emptyset$.

Cuando se realiza un revisión de núcleo (en inglés, *kernel revision*) por medio de un conjunto de sentencias, se necesita una función de incisión para realizar el corte en cada conjunto; *i.e.*, es necesario determinar qué creencias deben ser descartadas en el proceso de revisión. Hay dos políticas posibles: (1) descartar creencias particulares, y (2) descartar creencias generales. En la segunda, al menos una sentencias debe ser descartada. Falappa et al. (2002) proponen una caracterización refinada del proceso de revisión preservando creencias retractadas con un *status* diferente: las creencias generales retractadas son preservadas como *reglas rebatibles*. También introducen un operador de revisión que genera condicionales a partir de un operador de revisión que genera condicionales rebatibles a partir de bases de conocimiento representadas en un lenguaje de primer orden. Puede se el caso que, en el proceso de revisión, una sentencia de la forma $(\forall(X))(\alpha(X) \rightarrow \beta(X))$ tenga que ser eliminada. Esto puede ocurrir porque el agregado de nueva información resulte en una inconsistencia. Uno de los siguientes casos puede ocurrir: (1) existe un individuo satisfaciendo α pero no satisfaciendo β, y (2) existe un individuo satisfaciendo $\neg\beta$ pero no satisfaciendo $\neg\alpha$. La eliminación de $(\forall(X))(\alpha(X) \rightarrow \beta(X))$ de la base de conocimiento produciría mucha pérdida de información. Como una alternativa, Falappa et al. (2002) proponen una transformación para cambiarla en $\beta \prec \alpha$. Formalmente:

Definición 5.3.1 (Función de incisión externa (Falappa et al., 2002)) *Sea K un conjunto de sentencias. Una* función de incisión externa *para K es una función: $(\sigma) : 2^{2^{\mathcal{L}}} \mapsto 2^{\mathcal{L}}$, tal que para cada conjunto $A \subseteq \mathcal{L}$: (1) $\sigma((K \cup A)^{\perp\!\!\!\perp}\perp) \subseteq \bigcup((K \cup A^{\perp\!\!\!\perp}\perp)$, y (2) si $X \in (K \cup A)^{\perp\!\!\!\perp}\perp$ y $X \neq \emptyset$, entonces $(X \cap (K \cup A)^{\perp\!\!\!\perp}\perp) \neq \emptyset$.*

Definición 5.3.2 (Revisión de núcleo por medio de un conjunto de sentencias (Falappa et al., 2002)) *Sea K y A un conjunto de sentencias y (σ) una función de incisión externa para K. El operador $(\circ)$ de* revisión del núcleo por un conjunto de sentencias *$((\circ) : 2^{\mathcal{L}} \mapsto 2^{\mathcal{L}})$ se define como $K \circ A = (K \cup A) \setminus \sigma((K \cup A)^{\perp\!\!\!\perp}\perp)$.*

Definición 5.3.3 *Sea K un conjunto de sentencias y α una sentencia. Entonces $K^{\perp\!\!\!\perp}\alpha$ es el conjunto de todos los K' tales que $K' \in K^{\perp\!\!\!\perp}\alpha$ si y sólo si $K' \subseteq K$, $K' \vdash \alpha$, y si $K" \subset K'$ entonces $K" \nvdash \alpha$. El conjunto $K^{\perp\!\!\!\perp}\alpha$ se llama* conjunto núcleo, *y sus elementos se llaman los* α-núcleos de K.

Definición 5.3.4 (Transformación positiva/negativa (Falappa et al., 2002)) *Sea $\delta = (\forall X_1 \ldots X_n)(\alpha \rightarrow \beta)$ una implicación material en $\mathcal{L}^+$. Una* transformación positiva *de δ, notada como $T^+(\delta)$, es una sentencia de la forma $\beta \prec \alpha$; una* transformación negativa *de δ, notada como $T^-(\delta)$, es una sentencia de la forma $\neg\beta \prec \neg\alpha$.*

Definición 5.3.5 (Revisión de núcleo compuesta (Falappa et al., 2002)) *Sea (K, Δ) una estructura de conocimiento, $(\circ)$ un operador de revisión de núcleo por medio de un conjunto de sentencias para K y A como un conjunto de sentencias. La* revisión de núcleo

compuesta *de* (K, Δ) *con respecto a* A *se define como:* $(K, \Delta) \star A = (K', \Delta')$ *tal que* $K' = K \circ A$ *y* $\Delta' = \Delta \cup \Delta'_1 \cup \Delta'_2$ *donde:*

$$\begin{aligned} \Delta'_1 &= \{\ \alpha \prec true | \alpha \in (K_P \setminus K \circ A)\ \} \\ \Delta'_2 &= \{\ T^+(\alpha) | \alpha \in (K_G \setminus K \circ A)\ \} \cup \{\ T^-(\alpha) | \alpha \in (K_G \setminus K \circ A)\ \}. \end{aligned}$$

El conjunto K' contiene las reglas no rebatibles revisadas, Δ'_1 es la transformación en reglas rebatibles de creencias particulares (también llamadas *presumptions* (García and Simari, 2004, Sección 6)) eliminadas de K mientras que Δ'_2 es la transformación en reglas rebatibles de creencias generales eliminadas de K.

5.3.2. Mezcla de ontologías usando revisión de creencias

La *mezcla/integración* (en inglés, *merging/integration*) es el proceso de crear una nueva ontología a partir de dos o más ontologías existentes con partes solapadas (Klein, 2001). Adaptamos el marco de razonamiento con δ-ontologías presentado oportunamente en el Capítulo 4 para utilizarlo en la mezcla de ontologías.

En esta presentación, por simplicidad, asumimos restricción de nombres únicos, en el caso de no poder asumirse, es posible utilizar un esquema de integración de ontologías como el presentado en la Sección 4.6.

Definición 5.3.6 (δ-ontología mezclada) *Sean* Σ_1 *y* Σ_2 *dos* δ*-ontologías. La ontología mezclada entre* Σ_1 *y* Σ_2 *se denota como* $\Sigma_1 \oplus \Sigma_2$.

De la misma manera que con las δ-ontologías, la mezcla de dos δ-ontologías será interpretada como un programa DeLP.

Definición 5.3.7 (Interpretación de una δ-ontología mezclada) *Sean* Σ_1 *y* Σ_2 *dos* δ*-ontologías tales que* $\Sigma_1 = (T_S^1, T_D^1, A^1)$ *y* $\Sigma_2 = (T_S^2, T_D^2, A^2)$. *La interpretación de la* δ*-ontología mezclada, notada como* $\mathfrak{T}(\Sigma_1 \oplus \Sigma_2)$, *se define como el programa DeLP:*

$$\mathfrak{T}(\Sigma_1 \oplus \Sigma_2) = (\Pi, \Delta),$$

donde:

$$\begin{aligned} \Pi_1 &= \mathfrak{T}_\Pi(T_S^1) \cup \mathfrak{T}_\Pi(A^1); \\ \Delta_1 &= \mathfrak{T}_\Delta(T_D^1); \\ \Pi_2 &= \mathfrak{T}_\Pi(T_S^2) \cup \mathfrak{T}_\Pi(A^2); \\ \Delta_2 &= \mathfrak{T}_\Delta(T_D^2); \\ (\Pi, \Delta') &= (\Pi_1, \Delta_1) \star \Pi_2, \text{ y} \\ \Delta &= \Delta_1 \cup \Delta_2 \cup \Delta'. \end{aligned}$$

A continuación, extendemos las tareas de razonamiento sobre Aboxes para el caso de una ontología mezclada. En particular, definimos la operación de chequeo de instancia.

Definición 5.3.8 (Tareas de razonamiento en una δ-ontología mezclada) *Sean Σ_1 y Σ_2 dos δ-ontologías. Sea C un nombre de concepto y sea a un nombre de individuo.*

- *El individuo a pertenece potencialmente al concepto C respecto de la δ-ontología mezclada $\Sigma_1 \oplus \Sigma_2$ si y sólo si existe un argumento $\langle \mathcal{A}, C(a) \rangle$ respecto de la interpretación $\mathfrak{T}(\Sigma_1 \oplus \Sigma_2)$.*

- *El individuo a pertenece justificadamente al concepto C respecto de la δ-ontología mezclada $\Sigma_1 \oplus \Sigma_2$ si y sólo si existe un argumento garantizado $\langle \mathcal{A}, C(a) \rangle$ respecto de la interpretación $\mathfrak{T}(\Sigma_1 \oplus \Sigma_2)$.*

- *El individuo a pertenece estrictamente al concepto C respecto de la δ-ontología mezclada $\Sigma_1 \oplus \Sigma_2$ si y sólo si existe un argumento de la forma $\langle \emptyset, C(a) \rangle$ respecto de la interpretación $\mathfrak{T}(\Sigma_1 \oplus \Sigma_2)$.*

- *El individuo a pertenece en forma indeterminada al concepto C respecto de la δ-ontología mezclada $\Sigma_1 \oplus \Sigma_2$ si y sólo si no existe un argumento para el literal $C(a)$ respecto de la interpretación $\mathfrak{T}(\Sigma_1 \oplus \Sigma_2)$.*

Ejemplo 5.3.1 *Supongamos que tenemos la δ-ontología $\Sigma_1 = (T_S^1, T_D^1, A^1)$, donde:*

$$\begin{array}{rcl} T_S^1 & = & \left\{ \begin{array}{l} penguin \sqsubseteq bird; \\ bird \sqsubseteq flies \end{array} \right\}; \\ T_D^1 & = & \emptyset, \, y \\ A^1 & = & \left\{ \begin{array}{l} tweety : bird; \\ opus : penguin \end{array} \right\}. \end{array}$$

En este caso el conjunto de respuestas estrictas (el cual coincide con las justificadas puesto que la Dbox es un conjunto vacío) que obtenemos a partir de esta δ-ontología es

$$\mathfrak{StrictAnswers}(\Sigma_1) = \left\{ \begin{array}{l} bird(tweety), \\ penguin(opus), \\ bird(opus), \\ flies(tweety), \\ flies(opus) \end{array} \right\}.$$

Supongamos que recibimos otra ontología $\Sigma_2 = (T_S^2, T_D^2, A^2)$, vista como una explicación para "$opus : \neg flies$", donde:

$$\begin{array}{rcl} T_S^2 & = & \left\{ \, bird \sqcap penguin \sqsubseteq \neg flies \, \right\}, \, y \\ A^2 & = & \left\{ \begin{array}{l} opus : bird, \\ opus : penguin \end{array} \right\}. \end{array}$$

Ahora deseamos hallar el programa DeLP $\mathcal{P} = (\Pi, \Delta) = \mathfrak{T}(\Sigma_1 \oplus \Sigma_2)$ *interpretación de la* δ*-ontología mezclada entre* Σ_1 *y* Σ_2*. Así, cuando generamos la interpretación de la ontología mezclada, se debe realizar una revisión de núcleo por medio de un conjunto de sentencias. Necesitamos hallar los conjuntos mínimamente inconsistentes del conjunto de sentencias DeLP:*

$$\mathfrak{T}_\Pi(A^1) \cup \mathfrak{T}_\Pi(T_S^1) \cup \mathfrak{T}_\Pi(A^2) \cup \mathfrak{T}_\Pi(T_S^2).$$

Los dos conjuntos en estas condiciones son:

1. $\mathfrak{Transp}\left(\left\{ \begin{array}{l} bird(opus), \\ penguin(opus), \\ (flies(X) \leftarrow bird(X), penguin(X)), \\ (flies(X) \leftarrow bird(X)) \end{array} \right\}\right)$, *y*

2. $\mathfrak{Transp}\left(\left\{ \begin{array}{l} penguin(X), \\ (bird(X) \leftarrow penguin(X)), \\ (flies(X) \leftarrow bird(X)), \\ (\sim flies(X) \leftarrow bird(X), penguin(X)) \end{array} \right\}\right)$.

Para descartar creencias generales, debemos descartar al menos una sentencia en cada conjunto. Como la sentencia "$flies(X) \leftarrow bird(X)$*" está en ambos conjuntos, se la puede descartar. El conjunto de reglas estrictas* Π *de la base revisada está compuesta entonces por:*

$$\begin{array}{rcl} \Pi & = & \left\{ \begin{array}{l} bird(tweety), \\ bird(opus), \\ penguin(opus) \end{array} \right\} \cup \mathfrak{Transp}(\{\ bird(X) \leftarrow penguin(X)\ \}) \cup \\ & & \mathfrak{Transp}(\{\ \sim flies(X) bird(X), penguin(X) \leftarrow\ \}). \end{array}$$

En este caso, las consecuencias estrictas de la base revisada son:

$$\left\{\ bird(tweety), bird(opus), penguin(opus), \neg flies(opus)\ \right\}.$$

Sin embargo, la sentencias eliminadas no son completamente olvidadas sino que son almacenadas (transformadas) como reglas rebatibles. Esto es, el conjunto de reglas rebatibles Δ *de la interpretación de la* δ*-ontología mezclada es:*

$$\Delta \ = \ \left\{ \begin{array}{l} flies(X) \prec bird(X) \\ \sim bird(X) \prec\ \sim flies(X) \end{array} \right\}.$$

Entonces tenemos las siguientes conclusiones justificadas:

$$\mathfrak{JustifiedAnswers}(\Sigma_1 \oplus \Sigma_2) \quad = \quad \left\{ \begin{array}{l} bird(tweety), \\ bird(opus), \\ penguin(opus), \\ \neg flies(opus), \\ flies(tweety) \end{array} \right\}.$$

Note como el literal "flies(tweety)" se halla presente en el conjunto de conclusiones justificadas y no en el de conclusiones estrictas; i.e., *ahora además es posible afirmar que el individuo Tweety pertenece en forma justificada al concepto Flies.*

5.4. Resumen

En este capítulo, hemos analizado las propiedades formales que se desprenden de este acercamiento novedoso al tratamiento de ontologías inconsistentes en la Web Semántica. Los principales resultados obtenidos son que, como la interpretación de δ-ontologías como programas lógicos rebatibles es realizada a través de una función de transformación que preserva la semántica de las ontologías involucradas, los resultados obtenidos al consultar las operaciones sobre Aboxes son sensatas. Además, hemos mostrado que el operador presentado es además consistente y significativo.

Otros resultados mostrados están relacionados con una notación para extender la notación UML llamada δ-UML, la que permite caracterizar gráficamente algunos de los patrones de razonamiento que aparecen al representar conocimiento con δ-ontologías.

Además, hemos mostrado cómo la propuesta presentada puede ser extendida para implementar un operador de mezcla de ontologías. Este operador está basado en resultados tomados prestados del área de Revisión de Creencias. Así, mostramos cómo puede gestionarse la unión de la información terminológica estricta (o Sboxes) de las ontologías involucradas cuando la misma produce inconsistencia. El efecto final esta dado porque parte de la información de las Sboxes involucradas, que produce las inconsistencias indeseadas, es transformada para hacer las veces de información presentada en las Dboxes (o información terminológica tentativa).

A continuación, en el Capítulo 6, veremos cómo aplicar el marco desarrollado en el contexto de una aplicación web. En particular, mostraremos como el enfoque puede ser aplicado a un dominio bancario.

Capítulo 6

Caso de estudio: δ-Forms

En el Capítulo 4 vimos cómo es posible interpretar ontologías expresadas en un subconjunto particular de las Lógicas para la Descripción como un programa lógico rebatible con el objetivo de lidiar con ontologías inconsistentes. Dicho acercamiento al razonamiento en presencia de ontologías inconsistentes brinda la posibilidad de abordar de una manera eficaz ciertos problemas de aplicación del ámbito del comercio electrónico (en inglés, *e-commerce*), donde el modelo de reglas de negocio puede ser especificado en términos de ontologías. Entonces, la capacidad de razonar frente a ontologías inconsistentes permite abordajes alternativos conceptualmente más claros, ya que es posible automatizar ciertas decisiones de negocios tomadas a la luz de un conjunto de reglas de negocio posiblemente inconsistentes expresadas como una o varias ontologías y tener un sistema capaz de brindar una explicación del porqué se arribó a una conclusión determinada.

En este capítulo presentamos entonces una aplicación del razonamiento sobre ontologías inconsistentes por medio de la argumentación rebatible al modelado de formularios en la World Wide Web (Raggett et al., 1999). La noción de los formularios como una manera de organizar y presentar datos ha sido utilizada desde el comienzo de la World Wide Web (Raggett, 1995). Los formularios Web han evolucionado junto con el desarrollo de nuevos lenguajes de marcado, en los cuales es posible proveer guiones de validación como parte del código del formulario para verificar que si el significado pretendido del formulario es correcto (Raggett, 1997; Gulbransen and Rawlings, 1998). Sin embargo, para el diseñador del formulario, parte de este significado pretendido frecuentemente involucra otras características que no son restricciones por si mismas, sino más bien *atributos* emergentes del formulario, los cuales proveen conclusiones plausibles en el contexto de información incompleta y potencialmente contradictoria. Como el valor de tales atributos puede cambiar en presencia de nuevo conocimiento, los llamamos *atributos rebatibles*.

Proponemos extender los formularios web para incorporar atributos rebatibles como parte del conocimiento que puede ser codificado por el diseñador del formulario. Veremos cómo dicho conocimiento puede ser especificado mediante un programa DeLP, y posterior-

mente, como una ontología expresada en Lógicas para la Descripción. Así, la extensión propuesta permite la especificación de guiones para razonar acerca de los campos del formulario utilizando una base de conocimiento rebatible, expresada en términos de un Programa Lógico Rebatible. Los resultados presentados en este capítulo están parcialmente basados en los trabajos (Gómez et al., 2005*c*,*a*, 2008*b*,*d*).

6.1. Introducción y motivaciones

La noción de formulario como una manera de organizar y presentar datos es una abstracción estructural bien conocida para la recolección de datos, almacenamiento y recuperación de información. Los formularios son medios importantes para diseñar y desarrollar sistemas de información orientados al usuario, y han sido usados ampliamente desde el inicio de la World Wide Web. Los formularios web han evolucionado junto con la introducción de nuevos lenguajes de marcado (*e.g.*, XML), en el cual es posible proveer scripts de validación como parte de código del formulario para verificar si el significado intencional del formulario es correcto (Wu et al., 2004).

Como ya hemos explicado en capítulos anteriores de esta tesis, el cumplimiento de los objetivos del programa de la Web Semántica (Berners-Lee et al., 2001) requiere tener herramientas capaces de lidiar con las potenciales inconsistencias e incompletitudes de las fuentes de datos de la web (Guan and Mcmullen, 2005). Un dominio de aplicación particularmente importante es el de las tecnologías de comercio electrónico, las cuales típicamente demandan la validación de datos de usuario (*e.g.*, números de tarjetas de crédito) con respecto a un conjunto de criterios para determinar si un usuario dado es elegible para una cierta transacción posible. La realización de validaciones sobre los valores de los campos de un formulario permite determinar si el significado pretendido de tales campos es coherente de acuerdo a ciertos criterios establecidos por el diseñador del formulario. Tales validaciones usualmente consisten de un número de criterios de decisión fijos (en inglés, *hard-coded*), los cuales son especificados como una porción de código imperativo en un lenguaje de guiones (en inglés, *scripting language*).

Sin embargo, en muchos casos existen ciertas características emergentes las cuales pueden ser inferidas como parte del *"significado pretendido"* del formulario sin ser valores de campos. Así en el caso de aplicaciones de préstamos bancarios, la noción de *"cliente confiable"* aplicada sobre un cierto cliente particular puede ser inferida como una conclusión plausible a partir del conocimiento de ingreso anual y los registros bancarios de tal cliente. Tales características (o *atributos*) del formulario son difíciles de modelar en términos de piezas de código imperativo, particularmente en presencia de información incompleta y potencialmente contradictoria. Las conclusiones asociadas que pueden ser inferidas se tornan *rebatibles* ya que pueden cambiar a la luz de nueva información.

Nuestra propuesta consiste de extender los formularios web tradicionales para incorpo-

rar atributos adicionales como parte del conocimiento declarativo que puede ser codificado en un formulario. El valor de tales atributos dependerá de la consideración global de información incompleta y potencialmente contradictoria. Como estos valores puede cambiar en presencia de nueva evidencia o información, los llamaremos *atributos rebatibles*. La extensión propuesta permite la especificación de guiones (*scripts*) para manipular tales atributos rebatibles en la base de una base de conocimiento rebatible asociada al formulario, expresada en términos de de la *Programación en Lógica Rebatible* (García and Simari, 2004), una formalización particular de la argumentación rebatible (Chesñevar, Maguitman and Loui, 2000) basada en la programación en lógica. Mostraremos también como la Programación en Lógica Rebatible puede ser utilizada para modelar ontologías y también cómo esta extensión puede ser integrada con acercamientos existentes basados en el cliente (en inglés, *client-based)* para manipular formularios, como el uso de guiones de validación JavaScript. Finalmente, veremos cómo integrar la propuesta de extensión de formularios web con atributos rebatibles para que el significado de los mismos sea expresado en términos de δ-ontologías (que fueron presentadas en el Capítulo 4).

El resto del capítulo está estructurado como sigue: En la Sección 6.2, exploramos la utilización del lenguaje DeLP como herramienta de representación de conocimiento ontológico. En la Sección 6.3, presentamos una integración de formularios con argumentación rebatible, junto con un ejemplo que será utilizado para ilustrar los diferentes conceptos presentados en el capítulo; se describen aspectos genéricos acerca de los formularios web así como la importancia de introducir atributos rebatibles en formularios. En la Sección 6.4, mostramos cómo codificar formularios con atributos rebatibles usando XDeLP. En la Sección 6.5, discutimos cómo la redefinición de programas DeLP puede utilizarse para el refinamiento de ontologías. En la Sección 6.6, mostramos cómo la noción de formulario con atributos rebatibles puede ser integrada con los acercamientos corrientes en la Web Semántica al dar una caracterización de formularios con atributos rebatibles descriptos por medio de δ-ontologías. En la Sección 6.7, discutimos los trabajos relacionados en la literatura. Finalmente, en la Sección 6.8, presentamos las conclusiones de este capítulo.

6.2. Programas rebatibles como lenguaje de representación de conocimiento ontológico

El estándar para la Web Semántica requiere que las ontologías sean definidas en un lenguaje de definición de ontologías. Los lenguajes estándar como RDF, RDF-Schema y OWL-DL permiten cierto tipo de constructores, que fueron presentados oportunamente en el Capítulo 2. Sin embargo, hay ciertas construcciones que no son permitidas por OWL-DL. Parte de las restricciones impuestas por las construcciones disponibles de OWL-

DL pueden ser superadas al representar ontologías con un lenguaje de representación de conocimiento basado en la Programación en Lógica. Citamos textualmente a (Grosof et al., 2003):

> Las DLs son subconjuntos decidibles de la Lógica de Primer Orden donde la decidibilidad es debida en gran parte a que tienen la forma de la "propiedad de modelos en árbol". Esta propiedad dice que una clase DL C tiene un modelo (una interpretación $\mathcal{I}$ en la cual $C^{\mathcal{I}}$ es no vacía) si y sólo si C tiene un modelo en forma de árbol, *i.e.*, uno en la cual la interpretación de propiedades define un grafo dirigido con forma de árbol.
>
> Este requerimiento restringe severamente la forma en que las variables y los cuantificadores pueden ser usados. En particular, los cuantificadores deben ser "relativizados" a través de fórmulas atómicas (como en el fragmento con guardas de la Lógica de Primer Orden), *i.e.*, cada variable cuantificada debe ocurrir en un predicado de propiedad junto con la variable libre. Por ejemplo, la clase DL $\exists P.C$ corresponde a la fórmula de primer orden $\exists y.(P(x,y) \wedge C(y))$, donde el predicado P actúa como una guarda. Una obvia consecuencia de esta restricción es que es imposible describir clases cuyas instancias están relacionadas con otro individuo anónimo a través de caminos diferentes de propiedades. Por ejemplo, es imposible (en DL) expresar que los individuos que viven y trabajan en la misma ubicación son "trabajadores locales" (en inglés, *home workers*). Esto es fácil con una regla Horn, *e.g.*:
>
> $$home\_worker(x) \leftarrow work(x,y) \wedge live(x,y) \wedge loc(y,w) \wedge loc(z,w)$$
>
> Otra restricción en DLs es que sólo predicados unarios y binarios pueden usualmente ser capturados. Esta es una restricción menos onerosa, sin embargo, ya que las técnicas para reificar predicados de mayor aridad son bien conocidas.

Cuando las ontologías son inconsistentes, la representación de conocimiento basada en la Programación en Lógica puede ser mejorada utilizando la Programación en Lógica Rebatible. Considerando las dificultades de los enfoques actuales presentadas respecto de las deficiencias expresivas de las Lógicas para la Descripción mostradas, utilizaremos directamente a la Programación en Lógica Rebatible como formalismo alternativo de representación de ontologías.

Intuitivamente, una ontología es una colección de información, incluyendo generalmente información acerca de clases y propiedades. Por lo tanto, al considerar a un programa DeLP como una ontología es necesario tener en cuenta:

- un conjunto de reglas rebatibles para establecer la jerarquía de clases;

- un conjunto de reglas estrictas para establecer la jerarquía de clases;
- un conjunto de reglas rebatibles para establecer atributos de clases;
- un conjunto de reglas estrictas para establecer atributos de clases;
- un conjunto de reglas rebatibles para establecer restricciones sobre clases e instancias;
- un conjunto de reglas estrictas para establecer restricciones sobre clases e instancias;
- un conjunto de hechos para definir individuos de clases, y,
- un conjunto de relaciones entre individuos.

En los ejemplos subsiguientes veremos cómo representar cada uno de estos conjuntos.

Las *clases* corresponden a conjuntos de individuos y se definen mediante predicados unarios. Es decir, una clase C se representará con un predicado unario $C(X)$[1]. La noción de *subclase* relaciona una clase S más específica con una clase C. En nuestro acercamiento, consideraremos dos nociones de subclase: *subclase estricta* (representada con una regla estricta $C(X) \leftarrow S(X)$) y *subclase rebatible* (representada con una regla rebatible $C(X) \prec S(X)$). Es necesario notar que la decisión de cuál tipo de noción de subclase es necesario utilizar es puramente una decisión del ingeniero de conocimiento.

Ejemplo 6.2.1 *La idea que expresa que "absolutamente todos los individuos de la clase estudiante de doctorado son individuos de la clase persona"* (i.e., *la clase "phdstudent" es subclase estricta de la clase "student"), se representará con una regla estricta:*

$$student(X) \leftarrow phdstudent(X).$$

En cambio, la idea que establece que "usualmente todos los individuos de la clase estudiante de doctorado son individuos de la clase persona insolvente económicamente" (i.e., *la clase "phdstudent" es subclase rebatible de la clase "nonsolvent"), se representará con una regla rebatible:*

$$nonsolvent(X) \prec phdstudent(X).$$

La noción de *instancia* es la relación de pertenencia de un individuo a una clase. En nuestro acercamiento, la noción de individuo perteneciente a una clase está dada por un conjunto de hechos formado por predicados unarios fijos donde el nombre del predicado denota una clase y su argumento corresponde a una constante representando a un individuo en dicha clase.

[1]En este aspecto, nuestro enfoque coincide con los enfoques basados en las Lógicas para la Descripción (Baader et al., 2003).

Ejemplo 6.2.2 *Para representar la información que expresa que John es un estudiante de doctorado y que Mark es millonario usaremos los hechos pertenecientes al conjunto de individuos:*

$$phdstudent(john)$$
$$millionaire(mark)$$

Para denotar *relaciones* entre dos individuos de dos clases diferentes utilizaremos hechos binarios.

Ejemplo 6.2.3 *Para establecer que el padre de John es Eddie, usaremos el siguiente hecho:*

$$father(john, eddie)$$

El enfoque a la definición de ontologías que proponemos no limita el uso de relaciones a relaciones binarias; por el contrario, podemos utilizar relaciones n-arias para modelar relaciones generales.

Ejemplo 6.2.4 *Para establecer que un banco posee la información concerniente a que John vive en Krakosia y tiene un salario de $400 mensuales, tendremos el hecho:*

$$info(john, krakosia, 400)$$

El mundo de clases e individuos es muy limitado si sólo se pueden definir taxonomías. Los *atributos* permiten especificar hechos generales sobre los miembros de las clases y hechos específicos sobre los individuos. Separaremos los atributos en estrictos y rebatibles. También consideraremos el uso de reglas para especificar chequeo de tipos.

Ejemplo 6.2.5 *La clase "persona" posee los atributos nombre, edad y sexo de tipo string, entero y string respectivamente. Esta información se representa con las siguientes reglas:*

$$string(Y) \leftarrow person(X), name(X, Y)$$
$$integer(Y) \leftarrow person(X), age(X, Y)$$
$$string(Y) \leftarrow person(X), sex(X, Y)$$

También consideraremos *atributos emergentes* de la ontología que proveen conclusiones plausibles en presencia de información incompleta y potencialmente contradictoria. Como el valor de tales atributos puede cambiar en presencia de nuevo conocimiento, los llamaremos *atributos rebatibles*:

Ejemplo 6.2.6 *Para expresar que un estudiante de doctorado usualmente tiene un estado económico poco líquido a menos que su padre sea rico, usaremos las reglas:*

$$status(X, non\_liquid) \prec phdstudent(X)$$
$$status(X, liquid) \prec phdstudent(X), father(X, Y), rich(Y).$$

Una vez definidos las clases, los individuos, sus atributos y las relaciones entre individuos de clase, nos interesa poder especificar *restricciones* que permitan razonar sobre ellos y así enriquecer las conclusiones que se pueden obtener sobre los datos.

Ejemplo 6.2.7 *Para indicar que las nociones de liquidez e iliquidez son opuestas, usaremos la regla estricta:*

$$\sim status(X, liquid) \leftarrow status(X, non\_liquid)$$

Para indicar que una persona que no es solvente usualmente no tiene un buen ingreso, utilizaremos la regla:

$$\sim good\_income(X) \prec \sim solvent(X)$$

Los formularios Web han evolucionado para permitir el uso de guiones de validación como parte del código del formulario para verificar que si el significado pretendido del formulario es correcto (Raggett, 1997; Gulbransen and Rawlings, 1998). Sin embargo, para el diseñador del formulario, parte de este significado pretendido frecuentemente involucra otras características que no son restricciones por si mismas, sino más bien *atributos* emergentes del formulario, los cuales proveen conclusiones plausibles en el contexto de información incompleta y potencialmente contradictoria. Como el valor de tales atributos puede cambiar en presencia de nuevo conocimiento, los llamamos *atributos rebatibles*. Proponemos extender los formularios web para incorporar atributos rebatibles como parte del conocimiento que puede ser codificado por el diseñador del formulario. Veremos cómo dicho conocimiento puede ser especificado mediante un programa DeLP que define una ontología. En virtud de que la información contenida en la misma puede ser incompleta y potencialmente contradictoria, realizaremos un análisis dialéctico sobre las reglas y hechos que la conforman.

6.3. Integración de formularios con argumentación rebatible

La noción de formulario es una abstracción estructural central para la recolección de datos, el almacenamiento y la recuperación de datos en sistemas de gestión de información. Desde el comienzo mismo de la World Wide Web, los estándares HTML incluyeron la posibilidad de diseñar formularios, introduciendo un elemento de interactividad a un sitio web, junto con una variedad de técnicas de control de formularios. Los formularios proveen una manera estándar de permitir al usuario el envío de información de vuelta al servidor por medio de diferentes tecnologías para verificar y validar datos. Diversas tecnologías de programación fueron luego desarrolladas, permitiendo la creación de aplicaciones web interactivas que superaron en performance a las páginas web estáticas (Weber, 1997).

Por ejemplo HTML Dinámico (Gulbransen and Rawlings, 1998) favoreció el desarrollo de aplicaciones del lado del cliente al permitir embeber piezas de código de programas escritos en lenguajes de guión. La popularidad creciente de tecnologías de comercio electrónico (en inglés, *e-commerce*) así como la visión de la Web Semántica motivaron la especificación de estándares sofisticados para formularios web, notablemente *XForms*. Mostraremos cómo extender formularios web tradicionales para incluir conocimiento rebatible expresado en términos de un programa DeLP, caracterizando la noción de *formulario con atributos rebatibles*. Para hacer esto, primero presentaremos un caso de estudio del dominio de las aplicaciones bancarias que nos servirá para mostrar cómo codificar conocimiento correspondiendo a un problema del mundo real usando DeLP.

6.3.1. Caso de estudio: Los préstamos en un banco

A continuación, vamos a presentar un ejemplo en el dominio bancario para ilustrar nuestra propuesta.

Ejemplo 6.3.1 *Un banco internacional mantiene un seguimiento de sus clientes para determinar si otorga préstamos. Para cada cliente, el banco mantiene el nombre, país de origen, profesión, ingreso promedio mensual y el estado familiar del cliente. El gerente de cuentas del banco posee un número de criterios para otorgar préstamos. Los préstamos son otorgados si la persona tiene un "perfil" razonable, de acuerdo con sus registros personales. Las Figuras 6.1 y 6.2 muestran un programa DeLP $\mathcal{P}_{bank}$ para determinar el estado de tal solicitud de préstamo.*

Los hechos (1–3) de la forma $info(Name, Country, Profession, IncomePerMonth)$ describen información acerca de los clientes: el hecho (1) dice que John es un estudiante de doctorado de un país llamado Krakosia y tiene un ingreso promedio de \$400 *por mes; el hecho (2) dice que Ajax es también un estudiante de doctorado y tiene un ingreso promedio de* \$350 *por mes, y el hecho (3) dice que Danae es de Grecia con un ingreso de* \$10.000 *por mes y sin ninguna información acerca de su profesión. Los hechos (4–6) describen cuánto dinero ha sido solicitado por cada cliente al banco, mientras que los hechos (7–9) resumen los registros familiares de los clientes. Los hechos (10–11) establecen que Krakosia y Grecia son considerados como países confiables por las autoridades bancarias. El hecho (12) dice que Peter es uno de los gerentes del banco. Las reglas rebatibles (13–14) expresan que una persona P es usualmente candidata a recibir un préstamo si la persona P tiene el perfil indicado o si el préstamo pedido es razonable para el ingreso en 10 meses y P proviene de un país confiable. La regla (15) dice que un perfil confiable se define en términos del ingreso mensual y el país de origen. La regla (16) establece que usualmente todos los países son confiables. La regla (17) dice que una persona P tiene un ingreso razonable si éste es típicamente* \$300 *por mes o superior. La regla (18) expresa que usualmente un persona P quien no es económicamente solvente no tiene un ingreso razonable. Las reglas*

(19–20) dicen que usualmente los estudiantes de doctorado no son gente solvente a menos que provengan de familias adineradas. Además, la regla (21) dice que la gente catalogada por el banco con el estado "adinerado" se espera que provengan de familias adineradas. Finalmente, la regla (22) dice que los gerentes del banco son sin lugar a dudas candidatos para obtener préstamos.

Nótese que en este ejemplo en particular, tenemos que:

$$\mathsf{Pred}(\mathcal{P}_{bank}) = \left\{ \begin{array}{l} info/4, family\_record/2, req\_loan/2, credible/1, \\ candidate/1, profile\_ok/1, trustctry/2, goodincome/1, \\ solvent/1, richfamily/1, bankmanager/1 \end{array} \right\}.$$

Hechos (información provista por el usuario):
(1) $info(john, krakosia, phdstudent, 400)$.
(2) $info(ajax, greece, phdstudent, 350)$.
(3) $info(danae, greece, none, 10000)$.
(4) $req\_loan(john, 2000)$.
(5) $req\_loan(ajax, 4500)$.
(6) $req\_loan(danae, 1000)$.

Hechos (información del banco):
(7) $family\_record(john, rich)$.
(8) $family\_record(ajax, unknown)$.
(9) $family\_record(danae, unknown)$.
(10) $credible(krakosia)$.
(11) $credible(greece)$.
(12) $millionaire(peter)$.

Figura 6.1: Programa rebatible $\mathcal{P}_{bank}$ con criterios del banco para otorgar préstamos

Ejemplo 6.3.2 *Consideremos el programa DeLP mostrado en el ejemplo 6.3.1. Existe un argumento $\mathcal{A}$ apoyando la conclusión rebatible que John es un candidato para un préstamo, es decir, $\langle \mathcal{A}_1, candidate(john) \rangle$, donde:*

$$\mathcal{A}_1 = \left\{ \begin{array}{l} (candidate(john) \prec profile\_ok(john)), \\ (profile\_ok(john) \prec \\ \quad goodincome(john), trustctry(john, krakosia)), \\ (trustctry(john, krakosia) \prec \\ \quad info(john, krakosia, \_, \_), credible(krakosia)), \\ (goodincome(john) \prec info(john, \_, \_, 400), 400 > 300)\}, \end{array} \right\}$$

Otro argumento $\langle \mathcal{A}_2, \sim goodincome(john) \rangle$ puede ser derivado a partir de $\mathcal{P}_{bank}$, apoyando la conclusión que John no tiene un ingreso razonable, con:

Reglas rebatibles:
(13) $candidate(P) \prec profile\_ok(P)$.
(14) $candidate(P) \prec$
$info(P, \_, \_, Income), req\_loan(P, Amount),$
$Amount < Income * 10, trustctry(P, Ctry)$.
(15) $profile\_ok(P) \prec goodincome(P), trustctry(P, Ctry)$.
(16) $trustctry(P, Ctry) \prec info(P, Ctry, \_, \_), credible(Ctry)$.
(17) $goodincome(P) \prec info(P, \_, \_, Income), Income > 300$.
(18) $\sim goodincome(P) \prec \sim solvent(P)$.
(19) $\sim solvent(P) \prec info(P, \_, phdstudent, \_)$.
(20) $solvent(P) \prec info(\_, \_, phdstudent, \_), richfamily(P)$.
(21) $richfamily(P) \prec family\_record(P, rich)$.

Reglas estrictas:
(22) $candidate(P) \leftarrow millionaire(P)$.

Figura 6.2: Programa rebatible $\mathcal{P}_{bank}$ con criterios del banco para otorgar préstamos (cont.)

$$\mathcal{A}_2 = \left\{ \begin{array}{l} (\sim goodincome(john) \prec \sim solvent(john)), \\ (\sim solvent(john) \prec info(john, \_, phdstudent, \_)) \end{array} \right\}.$$

Usando la especificidad generalizada (Simari and Loui, 1992) como criterio de preferencia entre argumentos en conflicto, resulta que el argumento $\langle \mathcal{A}_2, \sim goodincome(john) \rangle$ *es un derrotador por bloqueo para el argumento* $\langle \mathcal{A}_1, candidate(john) \rangle$.

Ejemplo 6.3.3 *Consideremos la consulta* $candidate(john)$ *resuelta con respecto al programa* $\mathcal{P}_{bank}$ *(Figuras 6.1 y 6.2). Como se muestra en el Ejemplo 6.3.1, esta consulta comenzaría una búsqueda de argumentos apoyando al literal "*$candidate(john)$*", y encontraría al argumento* $\langle \mathcal{A}_1, candidate(john) \rangle$. *Para determinar si este argumento está garantizado, su árbol de dialéctica debe computarse: como se muestra en el Ejemplo 6.3.1, hay solamente un derrotador (por bloqueo) para el argumento* $\langle \mathcal{A}_1, candidate(john) \rangle$, *a saber,* $\langle \mathcal{A}_2, \sim goodincome(john) \rangle$. *Este derrotador, por su parte, tiene otro derrotador (propio)* $\langle \mathcal{A}_3, solvent(john) \rangle$, *con:*

$$\mathcal{A}_3 \;=\; \left\{ \begin{array}{l} (solvent(john) \prec info(john, \_, phdstudent, \_), richfamily(john)), \\ (richfamily(john) \prec family\_record(john, rich)) \end{array} \right\}$$

El árbol dialéctico resultante (marcado) se muestra en la Figura 6.3(i). Como el nodo del árbol dialéctico resultante es un nodo-U, la respuesta a $candidate(john)$ *es Sí.*

Consideremos ahora la consulta "$candidate(ajax)$*". Como en el caso de John, podemos encontrar el argumento* $\langle \mathcal{B}_1, candidate(ajax) \rangle$, *el cual es derrotado por el argumento* $\langle \mathcal{B}_2, \sim goodincome(ajax) \rangle$, *con:*

$$\mathcal{B}_1 = \left\{ \begin{array}{l} (candidate(ajax) \prec profile\_ok(ajax)), \\ (profile\_ok(ajax) \prec goodincome(ajax), trustctry(ajax, greece)), \\ (trustctry(ajax, greece) \prec info(ajax, greece, \_, \_), credible(greece)) \\ (goodincome(ajax) \prec info(ajax, \_, \_, 350), 350 > 300) \end{array} \right\},$$

$$\mathcal{B}_2 = \left\{ \begin{array}{l} (\sim goodincome(ajax) \prec \sim solvent(ajax)), \\ (\sim solvent(ajax) \prec info(ajax, \_, phdstudent, \_) \end{array} \right\}$$

Luego, el árbol dialéctico asociado para "candidate(ajax)" tiene dos nodos, con la raíz etiquetada como un nodo-D (vea la Figura 6.3(ii)). El argumento original para "candidate(ajax)" no está entonces garantizado. Finalmente, consideremos la consulta respecto del literal "candidate(danae)". Existe un argumento sin derrotadores (y, por lo tanto, garantizado) para esta consulta, como Danae tiene el perfil correcto para el banco:

$$\mathcal{C}_1 = \left\{ \begin{array}{l} (candidate(danae) \prec profile\_ok(danae)), \\ (profile\_ok(danae) \prec goodincome(danae), trustctry(danae, greece)), \\ (trustctry(greece) \prec info(danae, greece, \_, \_), credible(greece)), \\ (goodincome(danae) \prec info(danae, \_, \_, 10000), 10000 > 300) \end{array} \right\}$$

Ninguno de estos argumentos tiene derrotadores. Siguiendo el mismo razonamiento de arriba, ambos están garantizados. El árbol dialéctico resultante tendrá un único nodo, como se muestra en la Figura 6.3(iii).

Nótese que existe también un segundo argumento sin derrotadores apoyando la consulta "candidate(danae)", a saber $\langle \mathcal{C}_2, candidate(danae) \rangle$, con:

$$\mathcal{C}_2 = \left\{ \begin{array}{l} (candidate(danae) \prec \\ \qquad info(danae, \_, \_, 10000), req\_loan(danae, 1000), \\ \qquad 1000 < 10000 * 10, trustctry(danae, greece)), \\ (trustctry(danae, greece) \prec info(danae, greece, \_, \_), credible(greece)) \end{array} \right\}.$$

Finalmente, existe un argumento vacío $\mathcal{D}_1 = \emptyset$ —$\langle \emptyset, candidate(peter) \rangle$— apoyando la conclusión que Peter es un candidato para un préstamo pues es uno de los gerentes del banco. Como el conjunto de soporte de este argumento está compuesto de derivaciones estrictas, no tiene derrotadores (García and Simari, 2004), y por lo tanto está garantizado. El árbol dialéctico resultante está representado en la Figura 6.3(iv).

6.3.2. Formularios web: De HTML a XForms

La noción de formulario es una abstracción estructural central para la recolección de datos, el almacenamiento y la recuperación de datos en sistemas de gestión de información. Desde el comienzo mismo de la World Wide Web, los estándares HTML incluyeron

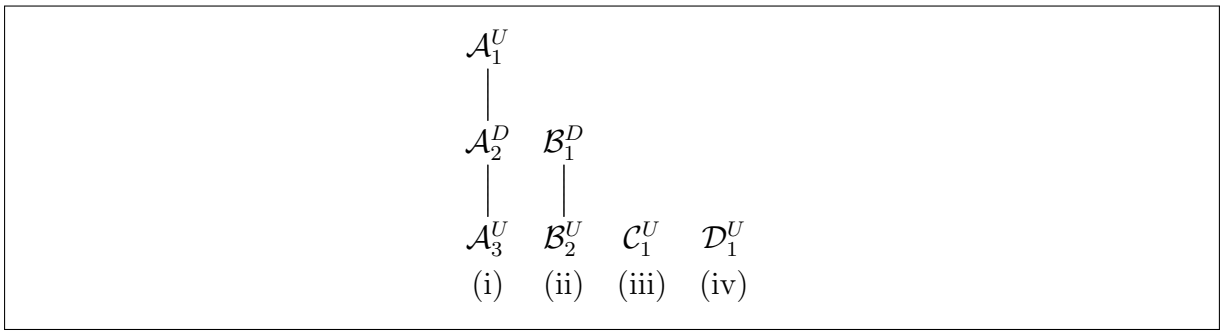


Figura 6.3: Árboles dialécticos para las consultas: (i) $candidate(john)$ con respecto a $\mathcal{P}_{bank}$; (ii) $candidate(ajax)$ con respecto a $\mathcal{P}_{bank}$, (iii) $candidate(danae)$ con respecto a $\mathcal{P}_{bank}$; (iv) $candidate(peter)$ con respecto a $\mathcal{P}_{bank}$

la posibilidad de diseñar formularios, introduciendo un elemento de interactividad a un sitio web, junto con una variedad de técnicas de control de formularios (cajas de edición y verificación, listas de selección, radio botones, botones, etc.). Los formularios proveen una manera estándar de permitir al usuario el envío de información de vuelta al servidor por medio de diferentes tecnologías para verificar y validar datos (por ej., guiones CGI (Dwight et al., 1997)). Un número de tecnologías de programación fueron luego desarrolladas (por ej., applets Java (Weber, 1997), HTML dinámico (Gulbransen and Rawlings, 1998)), permitiendo la creación de aplicaciones web interactivas que superaron en performance a las páginas web estáticas. Por ejemplo HTML Dinámico (en inglés, *Dynamic HTML* o simplemente *DHTML*) favoreció el desarrollo de aplicaciones del lado del cliente (en inglés, *client-side*) al permitir embeber piezas de código de programas escritos en lenguajes de guión (como JavaScript o VBScript). La popularidad creciente de tecnologías de comercio electrónico (en inglés, *e-commerce*) así como la visión de la Web Semántica motivaron la especificación de estándares sofisticados para formularios web, notablemente *XForms* (Dubinko et al., 2003), que combina la habilidad de separar propósito, presentación y resultados con XML en formularios web.

A pesar de la evolución de las tecnologías de los formularios basados en web, muchos diseñadores de formularios realizan la validación de campos de formularios por medio de forzar restricciones (*e.g.*, rangos numéricos) codificados como piezas de código imperativo en un lenguaje de guión (*e.g.*, JavaScript). Así, la validación de los datos es realizada en el lado del cliente, y los datos del formulario son finalmente procesados por un programa ubicado en un servidor remoto (usualmente accediendo algún tipo de base de datos). Sin embargo, en muchos casos existen ciertos atributos emergentes los cuales pueden ser inferidos como parte del "significado pretendido" del formulario sin ser valores de campos por si mismos. Así, en el caso de una aplicación de préstamos bancarios como la discutida en las secciones previas, un concepto como *cliente confiable*, modelado sobre la base de valores de campos para un cliente particular, podría ser útil desde el punto de

vista del diseñador del formulario para codificar criterios de toma de decisiones asociados con el procesamiento de formularios. Para identificar cada uno de los atributos relevantes necesarios para inferir un concepto como "cliente confiable" usando solamente código imperativo puede ser un tarea difícil, ya que en situaciones complejas tales conclusiones son rebatibles (particularmente en presencia de información incompleta y potencialmente contradictoria).

Para resolver el problema anterior, el concepto de formulario puede ser extendido apropiadamente para formilizar tales escenarios complejos sobre la base de DeLP por medio de *atributos rebatibles*, como veremos en la próxima sección. Para hacer esto, primero brindaremos una definición más bien genérica del concepto de formulario sobre la base de las nociones de *esquema de formulario* e *instancia de formulario*, la cual será útil para presentar nuestro acercamiento.

Definición 6.3.1 (Esquema de formulario. Instancia de formulario) *Un esquema de formulario es un par* $\mathcal{F} = \langle F, T \rangle$*, donde* $F = [f_1, f_2, \ldots, f_n]$ *es una lista de campos de formulario y* $T = [T_1, \ldots, T_n]$ *es una lista de tipos (cada uno de ellos consistiendo de un conjunto de valores). Dado un esquema de formulario* $\mathcal{F} = \langle F, T \rangle$ *definido como arriba, una instancia de formulario basada en* $\mathcal{F}$ *con valor* V *(denotada* $\mathcal{F}_V$*) es un par* $\mathcal{F}_V = \langle F, V \rangle$*, donde* $V = [v_1, \ldots, v_n]$ *es una lista de valores tales que cada* $v_i \in T_i$ *es el valor asociado para* $f_i \in F$*.*

Ejemplo 6.3.4 *Sean* F *= [name, profession, income, amountreq, country] y* T *= [string, string, float, integer, string], donde string, float e integer son nombres de tipos con el significado usual. Entonces* $\mathcal{F} = \langle F, T \rangle$ *es un esquema de formulario. Sea* V *= [john, phdstudent,* 400*,* 2000*, krakosia]. Entonces* $\mathcal{F}_V = \langle F, V \rangle$ *es una instancia de formulario basada en* $\mathcal{F}$*.*

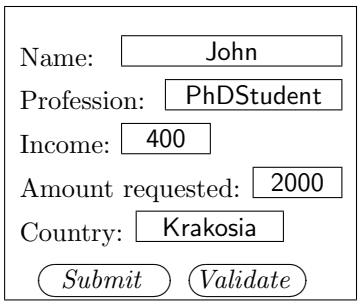


Figura 6.4: Vista de un formulario para la aplicación solicitudes de préstamos

La Figura 6.4 muestra la apariencia gráfica típica de un formulario web de acuerdo al esquema de formularios dado en el Ejemplo 6.3.4. Nótese que las acciones de control asociadas con el formulario —por ej., enviar datos (*submit*), restablecer (*clear*), etc— no son consideradas en la Definición 6.3.1.

6.3.3. Formularios con atributos rebatibles

En esta sección mostraremos cómo extender formularios web tradicionales para incorporar conocimiento rebatible expresado en términos de un programa DeLP, caracterizando la noción de *formularios con atributos rebatibles* o δ-formularios. Nuestro objetivo es el de proveer una manera de asociar un programa DeLP $\mathcal{P}$ con un esquema de formulario arbitrario (el cual se corresponderá con un número posiblemente diferente de instancias de formulario). El programa $\mathcal{P}$ se asume que representa conocimiento declarativo asociado con el dominio del problema del esquema de formulario. Así, como se discutió en el ejemplo precedente, una esquema de formulario correspondiente a una aplicación bancaria podría contener un programa DeLP asociado el cual representa políticas tentativas (y posiblemente conflictivas) para el otorgamiento de préstamos.

Dado un esquema de formulario $\mathcal{F} = \langle F, T\rangle$, y una instancia de formulario $\mathcal{F}_V$ particular, capturaremos el conocimiento factual involucrado en $\mathcal{F}_V$ en términos de un conjunto $facts(\mathcal{F}_V)$ de hechos DeLP. Tales hechos serán obtenidos por medio de la introducción de nuevos nombres de predicado asociados con aquellos nombres de campos en un formulario $\mathcal{F}$, y nuevos nombres de constantes correspondientes a los valores presentes en V. Formalmente:

Definición 6.3.2 (Conjunto de hechos $facts(\mathcal{F}_V)$) *Sea $\mathcal{F} = \langle F, T\rangle$ un esquema de formulario, con $F = [f_1, \ldots, f_n]$, y sea $\mathcal{F}_V$ una instancia de formulario. Definimos el conjunto de hechos $facts(\mathcal{F}_V)$ como:*

$$facts(\mathcal{F}_V) \;=_{def}\; \left\{\; f_1(\mathcal{F}, v_1), f_2(\mathcal{F}, v_2), \ldots, f_n(\mathcal{F}, v_n)\;\right\}.$$

Ejemplo 6.3.5 *Dada la instancia de formulario $\mathcal{F}_V$ en el Ejemplo 6.3.4, el correspondiente conjunto $facts(\mathcal{F}_V)$ es:*

$$facts(\mathcal{F}_V) \;=\; \left\{ \begin{array}{l} name(\mathcal{F}, john), \\ profession(\mathcal{F}, phdstudent), \\ income(\mathcal{F}, 400), \\ amountreq(\mathcal{F}, 2000), \\ country(\mathcal{F}, krakosia) \end{array} \right\}.$$

A continuación, mostraremos cómo los valores de los campos de un formulario web pueden ser integrados con un programa DeLP arbitrario $\mathcal{P}$, caracterizando el concepto de δ-formularios. Para ello utilizaremos la noción del conjunto de todos los nombres de predicados de un programa DeLP $\mathcal{P}$, notado como $\mathsf{Pred}(\mathcal{P})$.

Definición 6.3.3 (Esquema de formulario con atributos rebatibles. Instancia de δ-formulario) *Sea $\mathcal{F} = \langle F, T\rangle$ un esquema de formulario, y $\mathcal{P} = (\Pi, \Delta)$ un programa DeLP. Un esquema de formulario con atributos rebatibles (o esquema de δ-formulario) $\mathcal{D}$*

es un par $\langle \mathcal{F}, \mathcal{P} \rangle$. *Si* V *es un conjunto de valores para el formulario* $\mathcal{F}$, *una instancia de* δ*-formulario* $\mathcal{D}_V$ *es el par* $\langle \mathcal{F}_V, \mathcal{P} \rangle$. *El conjunto de atributos rebatibles para* $\mathcal{D}_V$ *se define como el conjunto de predicados* $\mathsf{Pred}(\Pi \cup \mathit{facts}(\mathcal{F}_V), \Delta)$.

Dado un esquema de δ-formulario $\langle \mathcal{F}, \mathcal{P} \rangle$, la definición anterior tiene como objetivo identificar características o atributos codificados por el diseñador del formulario como predicados distinguidos en el programa $\mathcal{P}$. Tales atributos se dicen *rebatibles*, pues su valor asociado puede ser determinado por consultas DeLP resueltas con respecto al programa DeLP $(\Pi \cup \mathit{facts}(\mathcal{F}_V), \Delta)$. Por lo tanto, un cambio en los valores de los campos en el formulario $\mathcal{F}$ o un cambio al programa DeLP subyacente $\mathcal{P}$ resultará en un cambio del valor para estos atributos. Como se estableció anteriormente, los atributos rebatibles representarán características relevantes para el diseñador de formularios, cuyos valores dependen del programa DeLP que codifica la parte relevante del conocimiento del dominio con la adición de hechos particulares que representan los valores de los campos para una instancia dada de formulario.

Ejemplo 6.3.6 *Sea* $\mathcal{F} = \langle F, T \rangle$ *el formulario presentado en el Ejemplo 6.3.4, y consideremos el programa:*

$$\mathcal{P}_{bank}{}' \;=\; \mathcal{P}_{bank} \setminus \{\ (1), \ldots, (6)\ \} \cup \{\ rule_1\ \} \cup \{\ rule_2\ \}$$

donde:

$$\begin{aligned} rule_1 \;&=\; info(N, C, P, I) \leftarrow \\ &\qquad\qquad name(\mathcal{F}, N), country(\mathcal{F}, C), \\ &\qquad\qquad profession(\mathcal{F}, P), income(\mathcal{F}, I) \\ rule_1 \;&=\; req\_loan(N, A) \leftarrow name(\mathcal{F}, N), amountreq(\mathcal{F}, A) \end{aligned}$$

es decir, el programa presentado en las Figuras 6.1 y 6.2 excluyendo la información provista por el usuario (reglas (1) a (6)), así como dos reglas estrictas adicionales $rule_1$ *y* $rule_2$ *enlazando el esquema de formulario* $\mathcal{F}$ *con las reglas del programa DeLP* $\mathcal{P}_{bank}$.

Sea $\mathcal{D} = \langle \mathcal{F}, \mathcal{P}_{bank}{}' \rangle$ *un esquema de* δ*-formulario. De acuerdo al programa DeLP* $\mathcal{P}_{bank}{}'$, *un atributo rebatible en* $\mathcal{D}$ *es* $candidate/1 \in \mathsf{Pred}(\mathcal{P}_{bank}{}')$. *Supongamos ahora que cuatro usuarios John, Ajax, Danae y Peter completan los campos de este* δ*-formulario como se describe en el Ejemplo 6.3.4. Por cada usuario* u, *se obtendría una instancia de* δ*-formulario particular* $\mathcal{D}_u$. *Así, al analizar, por ejemplo, la consulta "*$candidate(john)$*", la instancia de* δ*-formulario* $\mathcal{D}_{john}$ *involucraría la provisión de todos sus datos particulares como usuario, los cuales se harán presentes como un conjunto de hechos DeLP. Junto con las dos reglas estrictas adicionales presentadas anteriormente, razonando a partir de* $\mathcal{P}_{bank}{}' \cup \mathit{facts}(\mathcal{D}_{john})$ *resultaría en el análisis dialéctico mostrado en el Ejemplo 6.3.3, determinando que "*$candidate(john)$*" está garantizado. Lo mismo ocurre para*

los otros tres usuarios con respecto a las consultas "candidate(ajax)", "candidate(danae)" y "candidate(peter)", respectivamente.

En el ejemplo previo, hemos visto cómo los campos de formulario (ver la Definición 6.3.1) pueden ser referidos como nombres de predicados en DeLP. De forma similar, la noción de δ-formulario (ver la Definición 6.3.3) distingue algunos nombres de predicado como atributos rebatibles (como en el caso de *candidate*/1 mencionado previamente). Para usar tales atributos en el contexto de aplicaciones web, codificaremos a los programas DeLP en un lenguaje de marcado apropiado llamado X-DeLP.

6.4. X-DeLP: Codificación de DeLP en formularios web

En esta sección, presentaremos las características principales de *X-DeLP*, un lenguaje de guiones para codificar programas DeLP en XML. X-DeLP soporta la representación de bases de conocimiento rebatible por medio de la extensión de XHTML con marcadores que permiten representar programas lógicos rebatibles. X-DeLP puede embeberse directamente en documentos XHTML o usarse en documentos XML. Esta decisión de diseño provee varias ventajas (como remarcan Heflin *et al.* (Heflin et al., 2003) en el contexto de SHOE): (1) los autores web se sienten más confortables con la sintaxis XML pues existen muchas aplicaciones comerciales para editar documentos escritos en dicho formato; (2) su contenido de conocimiento puede ser usado en otras aplicaciones capaces de procesar XML; (3) el estándar de hojas de estilo en cascada (XSLT) (Clark, 1999) puede ser usado para para mostrar documentos XML en formas adecuadas para el entendimiento por usuarios humanos.

A continuación, explicaremos brevemente los elementos principales de la tecnología XML. Una discusión más profunda puede hallarse en (McGrath, 1998)

6.4.1. Fundamentos de XML

Los lenguajes de marcado basados en XML consisten de un conjunto de *tipos de elemento* que sirven para definir *tipos* de documentos y se hace referencia a los mismos como *Definiciones de Tipo de Documentos* o, simplemente, DTDs. A diferencia de HTML, donde los marcadores pertenecen a un conjunto estático, XML le permite a los autores crear sus propios marcadores (*e.g.*, <author>). Los documentos XML *bien formados* son documentos que satisfacen los requerimientos de la expecificación XML, mientras que los documentos *válidos* son aquellos que son bien formados y adicionalmente satisfacen todos los requerimientos especificados en el DTD. XML provee marcadores de inicio (*e.g.*, <foo>), marcadores de finalización (*e.g.*, </foo>), y marcadores vacíos (*e.g.*, <foo

bar="baz"/>). Los marcadores vacíos pueden tener *atributos* (*e.g.*, bar) que toman un *valor* (*e.g.*, baz). Los elementos que contienen alguna mezcla de marcadores y datos de tipo cadena de caracteres deben contener marcadores consistentes de inicio y final (*e.g.*, <country gov="democracy"> Krakosia </country>).

Las declaraciones de tipo de elemento permiten a una aplicación XML restringir los elementos que pueden ocurrir en el documento y también especificar el orden en que pueden ocurrir. La expresión <!ELEMENT foo EMPTY> declara elementos llamados foo como vacíos mientras que la expresión <!ELEMENT foo (apple|orange|banana)> declara que el elemento foo puede contener exactamente un elemento en el conjunto {apple, orange, banana}. Adicionalmente, XML permite declarar tipos de elementos que pueden contener otros elementos. La expresión <!ELEMENT person (name, address, phone?)> declara que una persona tiene un nombre, una dirección y, opcionalmente, un número de teléfono. Cero o más ocurrencias pueden ser especificadas como en <!ELEMENT books (book)*>, una o más ocurrencias como en <!ELEMENT books (book)+>. Los datos de tipo cadena de caracteres son denotados por la palabra clave #PCDATA como en <!ELEMENT quotation #PCDATA>. Las declaraciones de listas de atributos sirven para especificar el nombre, tipo y, opcionalmente, el valor por defecto de los atributos asociados con un elemento. Así, la expresión <!ATTLIST foo bar CDATA #REQUIRED> significa que el elemento foo tiene el atributo bar conteniendo datos de tipo cadena de caracteres que serán ignorados por el *parser* XML. El modificador #REQUIRED significa que es obligatorio dar un valor para el atributo. Otros modificadores como #IMPLIED son posibles indicando que el valor para el atributo será computado por una aplicación externa.

6.4.2. X-DeLP: Una caracterización sintáctica en XML

A continuación, presentaremos una caracterización de X-DeLP usando XML para ilustrar cómo las cualidades de DeLP pueden ser codificadas en formularios web. Para ejemplificar los conceptos que vamos a introducir, nuestra presentación estará basada en el programa $\mathcal{P}_{bank}$ presentado previamente en el Ejemplo 6.3.1.

Definición de un programa

Primero, se requiere un marcador <delp> para definir programas DeLP como entidades distinguidas en un documento XML. En nuestro caso de estudio el programa es llamado $\mathcal{P}_{bank}$, luego la representación XML será como a continuación:

<delp id="PBank" version="1.0">

Claramente, puede ser deseable anotar programas con comentarios, por ejemplo:

<comment>Program for defining criteria for granting a loan application.</comment>

Dentro de un esquema XML, un programa DeLP estará compuesto por definiciones de esquemas de reglas y declaraciones de instancias de reglas e instancias. La correspondiente representación se indica a continuación:

```
<!ELEMENT delp (comment?, definitions, declarations)>
<!ELEMENT comment PCDATA>
<!ATTLIST delp id CDATA #REQUIRED version CDATA #REQUIRED>
```

La definición de un programa involucra la especificación de los átomos y esquemas de reglas permitidos. Por ejemplo, para un definir un átomo *info*(*Name*, *Country*, *Profession*, *Income*), la representación DTD sería como sigue:

```
<def-atom name="info" arity="4">
        <def-arg pos="1" param="Name" type="string" />
        <def-arg pos="2" param="Country" type="string" />
        <def-arg pos="3" param="Profession" type="string" />
        <def-arg pos="4" param="Income" type="real" />
</def-atom>
```

Este DTD indica que el nombre de átomo es *info* con aridad 4. Además, el nombre y tipo de cada parámetro para *info* es especificado. La definición de átomos con aridad 0 es también posible, permitiendo la representación de programas DeLP proposicionales. La parte correspondiente del DTD se presenta a continuación:

```
<!ELEMENT definitions (def-language, def-rules)>
<!ELEMENT def-language (def-atom)*>
<!ELEMENT def-atom (def-arg)*>
<!ATTLIST def-atom name CDATA #REQUIRED arity CDATA #REQUIRED>
<!ELEMENT def-arg EMPTY>
<!ATTLIST def-arg pos CDATA #REQUIRED param CDATA #REQUIRED
        type CDATA #REQUIRED>
```

La reglas DeLP tendrán un único identificador y un tipo asociado (*defeasible* o *strict*). En particular, los argumentos (parámetros) literales pueden ser anónimos usando el calificador *dash*.[2] El atributo *negated* podría ser asociado con átomos en el programa. Un dado átomo A con nombre de predicado L que aparece com la cabeza de una regla en un programa puede tomar *yes* o *no* como valores posibles. En el primer caso, A representa $\sim L$ (*i.e.*, L es precedido por la negación estricta), mientras que en el segundo caso solamente representa L. El valor *no* es adoptado por defecto. Un análisis similar se aplica

[2]Note que esto corresponde a la variable anónima "guión bajo" en el lenguaje de programación Prolog.

para aquello átomos presentes en el cuerpo de una regla. Sin embargo, en este caso el atributo *negated* puede tomar tres valores posibles (*no*, *yes*, y *default*), los cuales representan L, $\sim L$, y *not* L (negación *default*), respectivamente. Las definiciones DTD se dan a continuación:

```
<!ELEMENT def-rules (def-rule)*>
<!ATTLIST def-rule id CDATA #REQUIRED
        type (defeasible | strict) #REQUIRED>
<!ELEMENT def-rule (comment?,def-head, def-body)>
<!ELEMENT def-head (arg)*>
<!ATTLIST def-head name CDATA #REQUIRED negated (no | yes) "no">
<!ELEMENT arg EMPTY>
<!ATTLIST arg pos CDATA #REQUIRED value (CDATA|dash) #REQUIRED>
<!ELEMENT def-body (def-body-atom)+>
<!ELEMENT def-body-atom (arg)*>
<!ATTLIST def-body-atom name CDATA #REQUIRED
        negated (no | yes | default) "no">
```

Ejemplo 6.4.1 *Considere el programa $\mathcal{P}_{bank}$ en las Figuras 6.1 y 6.2. La regla estricta (22) $candidate(P) \leftarrow millionaire(P)$ puede ser expresada como sigue:*

```
<def-rule id="22" type="strict">
        <def-head name="candidate" negated="no">
                <arg pos="1" param="P" type="string" />
        </def-head>
        <def-body>
                <def-body-atom name="millionaire" negated="no">
                        <arg pos="1" value="P" />
                </def-body-atom>
        </def-body>
</def-rule>
```

Análogamente, la regla (16) será expresada como:

```
<def-rule id="16" type="defeasible">
        <def-head name="trustctry" negated="no">
                <arg pos="1" param="P" type="string" />
                <arg pos="2" param="Ctry" type="string" />
        </def-head>
        <def-body>
                <def-body-atom name="info" negated="no">
```

```
                    <arg pos="1" value="P" />
                    <arg pos="2" value="Ctry" />
                    <arg pos="3" value="dash" />
                    <arg pos="4" value="dash" />
                </def-body-atom>
                <def-body-atom name="credible" negated="no">
                    <arg pos="1" value="Ctry" />
                </def-body-atom>
        </def-body>
</def-rule>
```

Definición de hechos y reglas

Dado un programa DeLP $\mathcal{P}$, las definiciones DTD previas representan una formulación basada en XML de $\mathcal{P}$. Sin embargo, el modelado del concepto de argumento requiere la *instanciación fija* de reglas. Las definiciones DTD se muestran a continuación:

```
<!ELEMENT declarations (rule-instances, facts)>
<!ELEMENT rule-instances (rule-instance)*>
<!ATTLIST rule-instance id CDATA #REQUIRED>
<!ELEMENT rule-instance (subst)*>
<!ELEMENT subst EMPTY>
<!ATTLIST subst param CDATA #REQUIRED value CDATA #REQUIRED>
<!ELEMENT facts (fact)*>
<!ELEMENT fact (arg)*>
<!ATTLIST fact negated (yes | no) "no"
        type (fact | assumption) "fact" name CDATA #REQUIRED>
```

Ejemplo 6.4.2 *Los hechos corresponden naturalmente a información fija. Así, utilizando nuestra representación DTD, el hecho que John es un estudiante de doctorado proveniente de Krakosia y que tiene un ingreso de* \$400 *($info(john, krakosia, phdstudent, 400)$) será codificado como:*

```
<fact negated="no" type="fact" name="info">
        <arg pos="1" value="john"/>
        <arg pos="2" value="krakosia"/>
        <arg pos="3" value="phdstudent"/>
        <arg pos="4" value="400"/>
</fact>
```

Análogamente, para representar la instancia de la regla:

$$trustctry(john, krakosia) \prec info(john, krakosia, \_, \_), credible(krakosia),$$

escribiremos:

```
<rule-instance id="16">
        <subst param="P" value="john" />
        <subst param="Ctry" value="krakosia" />
</rule-instance>
```

Definición de argumentos y árboles de argumentos

Finalmente, se proveen marcadores apropiados para representar en X-DeLP argumentos, líneas de argumentación y árboles dialécticos. Un argumento $\langle \mathcal{A}, H \rangle$ será representado por un conjunto $\mathcal{A}$ de instancias de reglas y una instancia de un hecho H, a saber:

```
<argument>
        <rule-instances>... representación XML para A...</rule-instances>
        <fact>... representación XML para H...</fact>
</argument>
```

Para representar el conjunto de todas las reglas (estrictas y rebatibles) usadas para la derivación de un argumento, también se provee un marcador llamado <complete-argument>.[3] Para representar el árbol que soporta la derivación de un argumento, también se provee un marcador llamado <argument-derivation>. Finalmente, se proveen marcadores para representar líneas de argumentación y árboles de argumentos. Cada nodo de un árbol de argumento tiene asociado un estado epistémico que puede tomar uno de dos valores: derrotado o no derrotado. A continuación, presentamos las definiciones DTD correspondientes:

```
<!ELEMENT argument (rule-instances, fact)>
<!ELEMENT complete-argument
        (rule-instances, facts, fact)>
<!ELEMENT argument-derivation (fact, argument-derivation*)>
<!ELEMENT argument-line (argument)*>
<!ELEMENT argument-tree (argument, argument-tree*)>
<!ATTLIST argument-tree epistemic-status (defeated | undefeated) #IMPLIED>
```

[3]Nótese que la definición de argumento solamente requiere considerar un conjunto finito de instancias fijas de reglas rebatibles. Sin embargo, para propósitos de implementación, puede ser útil mantener todas las reglas (estrictas y rebatibles) usadas en la construcción de un argumento (Chesñevar, Simari and Godo, 2005).

6.4.3. Embebiendo DeLP en el código del lado del cliente

Como mencionamos anteriormente, los programadores usualmente validan datos de formularios por medio de código imperativo asociado a eventos relacionados con cambios en los controles del formulario (botones, radio botones, casillas de verificación, listas de selección, etc.) y embebido en el código del documento web como un guión JavaScript. Tales piezas de código típicamente permiten validar valores de campos, permitiendo determinar si el significado pretendido de tales campos es coherente de acuerdo a los criterios establecidos por el diseñador del formulario. Sin embargo, como hemos visto en las secciones precedentes, existen muchos casos en los cuales puede ser útil incluir como parte de dichas validaciones algunas características que puede ser inferidas como parte del "significado pretendido" del formulario sin ser valores de campos por si mismos.

Sobre la base de la formalización de X-DeLP presentada en la Sección 6.4, proponemos integrar el motor de inferencia DeLP con un navegador web (en inglés, *web browser*), extendiendo el lenguaje de programación JavaScript [4] con primitivas adecuadas. Mostramos la arquitectura del enfoque en la Figura 6.6. La extensión a JavaScript consiste de primitivas para invocar los servicios del motor de inferencia DeLP. Esto es implementado a través de mensajes booleanos como[5]:

bool warranted(ProgID, Query).

Esta primitiva recibe dos parámetros adicionales que son el identificador del programa rebatible $ProgID$ y el literal consultado $Query$. La primitiva *warranted* determina si existe un literal $Query$ garantizado con respecto al programa DeLP $ProgID$. Funciones similares son implementadas para atributos rebatibles, por ejemplo:

bool undecided(ProgID, Query).

A continuación, mostraremos un ejemplo de cómo el acercamiento propuesto trabaja en un guión JavaScript ubicado en el lado del cliente.

Ejemplo 6.4.3 *Supongamos que $\mathcal{P}$ es un formulario rebatible (δ-form) como se describió en el Ejemplo 6.3.4, el cual por su parte está escrito en XForms y tiene a $form1$ como su identificador. Entonces, un programador JavaScript sería capaz de escribir el siguiente código embebido en una función manejadora para el botón de pulso Validate como se muestra en la Figura 6.5.*

[4]Por simplicidad, nos restringimos al caso de JavaScript para nuestro análisis. El acercamiento puede ser naturalmente extendido para cualquier otro lenguaje de guiones.

[5]Nótese que este enfoque coincide con la propuesta del *Document Object Model* (Gulbransen and Rawlings, 1998).

```
<script language="JavaScript">
        function validate()
        {
                if( form1.warranted(pbank, candidate(form1.name.value)) )
                    alert( "The requested loan will be probably granted." +
                           "We will contact you in a week." );
                else
                    alert( "Your case will be analyzed and " +
                           "we will contact you in a month." );
        }
</script>
```

Figura 6.5: Un ejemplo de una función JavaScript con capacidad de acceso al motor DeLP asociada a un botón de un formulario

6.5. Redefinición de programas rebatibles

Una manera común de proveer semántica a los documentos de la Web es a través del uso de *ontologías*. Como vimos en el Capítulo 2, en ciencias de la computación, las ontologías establecen una terminología conjunta entre los miembros de una comunidad de interés (*e.g.*, entre humanos o agentes automatizados), las cuales incluyen definiciones interpretables por máquina de conceptos básicos en el dominio y las relaciones que se cumplen entre dichos objetos. Las ontologías son usualmente expresadas en lenguajes basados en lógica, ya que proveen una manera declarativa de expresar conocimiento junto con capacidades de inferencia para razonar sobre los conceptos representados. En este contexto X-DeLP ofrece una alternativa interesante para modelar ontologías (usando definiciones de predicados) y la de realizar inferencias sobre la base del motor argumentativo subyacente. En un marco ontológico, un programa DeLP $\mathcal{P}$ (ver la Definición 3.3.4) puede también pensarse como un conjunto de definiciones de predicados; es decir, $\mathcal{P} =_{def} \{P_1^{\mathcal{P}}, \ldots, P_k^{\mathcal{P}}\}$. Así, en nuestro ejemplo concerniente a los préstamos bancarios, el programa $\mathcal{P}_{bank}$ (véase las Figuras 6.1 y 6.2) provee la definición de un número de predicados (*candidate*, *trustctry*, etc.). En un trabajo previo (Gómez et al., 2005*a*), mostramos cómo esta idea puede ser orientada para proveer un primer enfoque para formalizar *ontologías basadas en argumentación* en las cuales X-DeLP podría ser usado para el intercambio de ontologías en el contexto de la Web Semántica.

La formalización de técnicas para la *mezcla* y el *refinamiento* de ontologías (Noy, 2004; Dou et al., 2005) es una área activa de investigación en varias disciplinas (tales como bases de datos deductivas e integración de información). En particular, las ontologías pueden ser sujetas a *refinamientos* o cambios a la luz de nueva información. Para mostrar nuestro

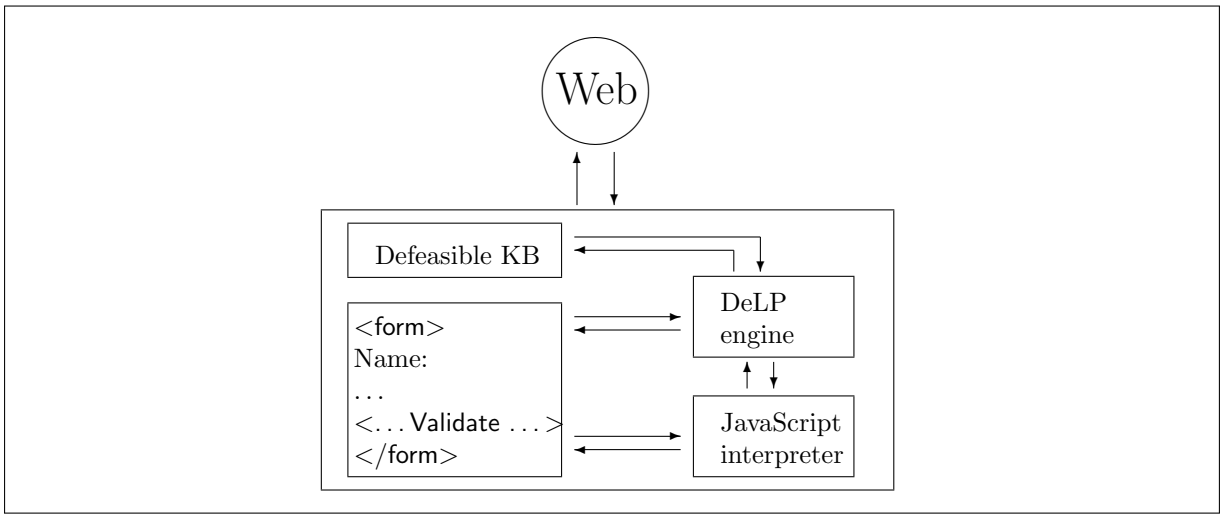


Figura 6.6: Un marco para embeber el motor de inferencias DeLP en un *browser*.

acercamiento, mostraremos un enfoque ingenuo de cómo X-DeLP puede ser usado para modelar tales cambios por medio de la noción de *redefinición de programa*, un concepto tomado de implementaciones de la programación en lógica[6]. Una redefinición de un programa $\mathcal{P}_1$ con respecto a otro programa $\mathcal{P}_2$ es un nuevo programa DeLP que incluye todas las definiciones de predicados en $\mathcal{P}_1$ y $\mathcal{P}_2$, excepto por aquellos predicados en $\mathcal{P}_1$ que también están presentes en $\mathcal{P}_2$. Formalmente:

Definición 6.5.1 (Redefinición) *Sean $\mathcal{P}_1, \mathcal{P}_2$ dos programas DeLP, tal que $\mathcal{P}_1$ define los predicados R_1, R_2, ..., R_n, y $\mathcal{P}_2$ define los predicados S_1, ..., S_m. La redefinición de $\mathcal{P}_1$ con respecto a $\mathcal{P}_2$, denotada $\mathcal{P}_1 \triangleleft \mathcal{P}_2$, es un nuevo programa $\mathcal{P}'$ definido como sigue:*

$$\begin{aligned}
\mathcal{P}' \quad &= \quad \mathcal{P}_1 \triangleleft \mathcal{P}_2 \\
&=_{def} \quad \left\{ \; R_1^{\mathcal{P}_1}, \ldots, R_n^{\mathcal{P}_1} \; \right\} \cup \left\{ \; S_1^{\mathcal{P}_2}, \ldots, S_m^{\mathcal{P}_2} \; \right\} \setminus \\
&\qquad \left\{ \; R_i^{\mathcal{P}_1} \mid \exists S_j^{\mathcal{P}_2} \text{ en } \mathcal{P}_2, \text{ con } name(R_i) = name(S_j), \text{ y } arity(R_i) = arity(S_j) \; \right\}.
\end{aligned}$$

De esta manera, la redefinición de un programa DeLP básicamente involucra la provisión de nuevas definiciones de predicados, que suplantan definiciones existentes (en el caso en que las hubiera).

Supongamos que el banco obtiene un número de criterios obtenidos de la Oficina de Seguridad de los Estados Unidos (en inglés, la *Homeland Security Office* o *HSO*) con respecto a cómo evaluar la honradez o confiabilidad de los distintos países. Tales criterios podrían estar codificados por los programadores de la HSO en DeLP como se muestra en la Figura 6.7: se tiene información acerca de Grecia y Krasosia como países con gobiernos

[6]En muchas implementaciones de Prolog, el resultado de la redefinición del programa corriente $\mathcal{P}_1$ con respecto a otro programa $\mathcal{P}_2$ puede ser modelado con el comando `reconsult(`$\mathcal{P}_2$`)`.

democráticos, y el hecho que Krakosia es un país en guerra. Las reglas rebatibles proveen criterios tentativos para rotular a un país como "creíble": los países democráticos son creíbles, a menos que estén en guerra o tengan gobiernos corruptos. Las autoridades bancarias podrían por lo tanto mezclar sus base de conocimiento $\mathcal{P}_{bank}$ considerando la información dada en $\mathcal{P}_{sec}$. El programa redefinido $\mathcal{P}_{bank} \lhd \mathcal{P}_{sec}$ resultante consideraría un análisis más detallado para los países, pues factores tales como sistema político, situación política, etc. serían tenidos en cuenta para conceder préstamos, como se muestra en el siguiente ejemplo.

Ejemplo 6.5.1 *Considere los programas DeLP* $\mathcal{P}_{bank}$=$\{(1), \ldots, (22)\}$ *y* $\mathcal{P}_{sec}$=$\{(1'), \ldots, (6')\}$ *de las Figuras 6.1 y 6.2, y 6.7, respectivamente. A través del cálculo de* $\mathcal{P}_{bank} \lhd \mathcal{P}_{sec}$ *obtenemos como resultado un nuevo programa DeLP:*

$$\mathcal{P}' \;=\; \{\ (1), \ldots, (22)\ \} \cup \{\ (1'), \ldots, (6')\ \} \setminus \{\ (10), (11)\ \},$$

en el cual la definición de "credible" provista por $\mathcal{P}_{bank}$ *es reemplazada por la nueva definición dada en* $\mathcal{P}_{sec}$*. La resolución de la consulta "*$candidate(john)$*" con respecto a* $\mathcal{P}'$ *involucra buscar argumentos similares a aquellos encontrados en el Ejemplo 6.3.3: un argumento* $\langle \mathcal{A}'_1, candidate(john) \rangle$ *soporta el literal "*$candidate(john)$*",*[7] *con:*

$$\mathcal{A}'_1 \;=\; \left\{ \begin{array}{l} (candidate(john) \prec profile\_ok(john)), \\ (profile\_ok(john) \prec goodincome(john), trustctry(john, krakosia)), \\ (trustctry(krakosia) \prec info(john, krakosia, \_, \_), credible(krakosia)), \\ (credible(krakosia) \prec country(krakosia, democracy)), \\ (goodincome(john) \prec info(john, \_, \_, 400), 400 > 300) \end{array} \right\}.$$

Como en el Ejemplo 6.3.3, este argumento es derrotado por otro argumento $\mathcal{A}_2$ *a favor de John,* $\langle \mathcal{A}_2, \sim goodincome(john) \rangle$*, el cual por su parte es derrotado por otro argumento* $\langle \mathcal{A}_3, solvent(john) \rangle$*. En todos estos argumentos, sin embargo, el programa redefinido permite inferir un cuarto argumento, a saber:* $\langle \mathcal{A}_4, \sim credible(krakosia) \rangle$ *con:*

$$\mathcal{A}_4 \;=\; \left\{ \begin{array}{l} \sim credible(krakosia) \prec \\ \qquad country(krakosia, democracy), country(krakosia, atwar) \end{array} \right\},$$

el cual es un derrotador propio para $\langle \mathcal{A}'_1, candidate(john) \rangle$*. Como resultado, la raíz del arbol de dialéctica para la consulta "*$candidate(john)$*" es marcado como un nodo-D (derrotado), como muestra en la Figura 6.8.*

Nótese que la redefinición de un programa usualmente resultará en el suministro de información *más específica* asociada con predicados particulares. Así, los argumentos en un programa $\mathcal{P}_1$ que tenían un estado epistémico particular (por ej., garantizados o, en inglés, *warranted*) pueden perderlo en un versión redefinida $\mathcal{P}_1 \lhd \mathcal{P}_2$.

$(1')$ $country(greece, democracy)$
$(2')$ $country(krakosia, democracy)$
$(3')$ $country(krakosia, atwar)$
$(4')$ $credible(Ctry) \prec country(Ctry, democracy).$
$(5')$ $\sim credible(Ctry) \prec$
$country(Ctry, democracy),$
$country(Ctry, atwar).$
$(6')$ $\sim credible(Ctry) \prec$
$country(Ctry, democracy),$
$country(Ctry, corruptgovt).$

Figura 6.7: Programa lógico rebatible $\mathcal{P}_{sec}$ obtenido de la *Oficina de Seguridad Interna* (HSO) de los Estados Unidos

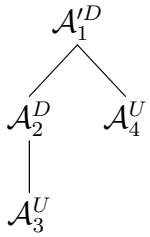


Figura 6.8: Árbol dialéctico para la consulta $candidate(john)$ con respecto a $\mathcal{P}_{bank} \triangleleft \mathcal{P}_{sec}$

A continuación presentamos algunas propiedades del operador de redefinición de programas rebatibles.

Propiedad 6.5.1 *El programa vacío es el neutro a derecha del operador de redefinición* $\triangleleft$.

Observación 6.5.1 *El operador de redefinición* $\triangleleft$ *no es en general asociativo.*

6.6. Integración de δ-formularios con Lógicas para la Descripción

6.6.1. Ontologías en aplicaciones comerciales

En las aplicaciones de las tecnologías de la Web Semántica al comercio electrónico (Corcho et al., 2003; Fensel et al., 2002), en particular para el soporte a la toma de deci-

[7]Nótese que el argumento $\langle \mathcal{A}_1', candidate(john) \rangle$ que involucra información rebatible acerca de Krakosia proveniente de $\mathcal{P}_{sec}$, en contraste con el argumento original $\langle \mathcal{A}_1, candidate(john) \rangle$.

siones (Smirnov et al., 2005), un desafío consiste de combinar el conocimiento operativo, basado en la experiencia profesional, con el conocimiento formal contenido en la legislación o en los manuales de procedimientos relevantes a la tarea siendo considerada. Además, es necesario poder tener una herramienta que pueda ser creada y reconfigurada rápidamente para dar soporte a servicios a clientes en un mundo de cambio constante. En particular, esta necesidad también se presenta en el contexto de los δ-formularios donde un conjunto de criterios para clasificar a usuarios como parte de un concepto descripto como un conjunto de atributos rebatibles pueden ser especificados por medio de una ontología.

Wagner (Wagner, 2003) puntualiza la necesidad de poder representar información negativa en las aplicaciones de la Web Semántica. En este respecto, los desarrollos recientes asociados al razonamiento en la web proponen el uso de lenguajes basados en Lógicas para la Descripción muy expresivas (DL) (Baader et al., 2003) (tales como DAML (McGuiness et al., 2003), OIL (Decker et al., 2000), DAML+OIL (Connolly et al., 2001) y OWL (McGuiness and van Harmelen, 2004)) para representar ontologías. A pesar de que tales acercamientos combinan satisfactoriamente factores de expresividad y eficiencia, fallan en lidiar con bases de conocimiento inconsistentes ya que los sistemas existentes sólo realizan un chequeo de consistencia mientras dejan el trabajo de eliminar las inconsistencias al programador.

En este sentido, en capítulos anteriores, mostramos cómo las δ-ontologías se presentan como una alternativa válida para poder manipular y razonar en presencia de ontologías posiblemente inconsistentes expresadas en Lógicas para la Descripción. La separación de las definiciones ontológicas en dos conjuntos — el conjunto de las definiciones estrictas o *Sbox* y el conjunto de las definiciones rebatibles o *Dbox* — permite separar el conjunto de la información válida de la tentativa y posiblemente contradictoria. Por ello, surge la necesidad de poder combinar los δ-formularios presentados anteriormente con las tecnologías estándar de la Web Semántica; pero, debido a que los criterios asociados a un δ-formulario pueden contener inconsistencias, y, como éstas no pueden ser manipuladas satisfactoriamente en el contexto de las tecnologías de representación de conocimiento estándar, presentaremos una forma de integración de δ-ontologías para modelar atributos rebatibles en el contexto de los δ-formularios.

6.6.2. Extensión de las δ-ontologías para representar dominios concretos

En la Sección 2.3.4, explicamos brevemente los acercamientos de la literatura de las Lógicas para la Descripción para representar ontologías con clases cuyas instancias poseen atributos con dominios concretos. Dichas extensiones son necesarias a la hora de poder representar por ejemplo el nombre de un cliente que pide un préstamo, el monto pedido o el ingreso mensual del cliente. Por ello, en esta sección extenderemos apropiadamente la

función de traducción de Lógicas para la Descripción en Programación en Lógica Rebatible para considerar el dominio concreto de los números enteros. Más adelante, veremos cómo esta adaptación nos permitirá representar los ingresos de una persona que llena un formulario así como el monto de préstamo requerido por un cliente.

Para ello sólo consideraremos los operadores relacionales de comparación por igual ($=$), distinto ($\neq$), mayor ($>$), mayor o igual ($\geq$), menor ($<$) y menor o igual ($\leq$) así como los predicados suma ($+$), resta (-), producto (*) y división ($/$). Como todos los casos considerados previamente en la Sección 4.2, nuestro objetivo es la representación de expresiones DL en el marco de la programación en lógica, en particular en la Programación en Lógica Rebatible. Por esta causa, las comparaciones y las operaciones aritméticas tendrán solamente sentido si aparecen en el miembro izquierdo de una sentencia de inclusión DL; este tendrá como efecto que la traducción a programación en lógica de dicha expresión aparezca en el cuerpo de una cláusula de Horn. En caso contrario, *i.e.* si la comparación o predicado asociado a una operación aritmética apareciera en el miembro derecho de una sentencia de inclusión, la traducción a programación en lógica no tendría sentido pues la comparación o cálculo aritmético aparecería en la cabeza de una cláusula de Horn.

Definición 6.6.1 (Lenguaje $\mathcal{L}_b$. $\mathcal{L}_b$-clases (versión extendida)) *Sea A un nombre atómico de clase, C y D expresiones de clases, R una propiedad y f un atributo funcional. Sea $\mathbb{N}$ el conjunto de los números naturales. Sean $op \in \{=, \neq, >, \geq, <, \leq\}$ y $n \in \mathbb{N}$. En el lenguaje $\mathcal{L}_b$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$ y $\exists f.op_n$ son expresiones de clases. Las expresiones de clases en el lenguaje $\mathcal{L}_b$ se llamarán $\mathcal{L}_b$-clases.*

Como también explicamos en el capítulo 4, la semántica de una δ-ontología está definida en términos de un programa DeLP. En la Definición 4.3.7 vimos cómo transformar una Dbox en un programa DeLP. Ahora extendemos tal definición para considerar dominios concretos. Formalmente:

Definición 6.6.2 (Correspondencia $\mathfrak{T}_\Delta$ desde sentencias DL hacia reglas rebatibles DeLP (versión extendida)) *Sean A, C, D conceptos, X, Y variables, P, Q relaciones, f un nombre de atributo, n un número entero. La correspondencia $\mathfrak{T}_\Delta : 2^{\mathcal{L}_{DL}} \rightarrow 2^{\mathcal{L}_{DeLP_\Delta}}$ presentada en la Figura 4.2 se extiende con las siguientes ecuaciones:*

$$
\begin{aligned}
T_b((\exists f. =_n) &=_{def} f = n \\
T_b((\exists f. \neq_n) &=_{def} f \backslash= n \\
T_b((\exists f. >_n) &=_{def} f > n \\
T_b((\exists f. \geq_n) &=_{def} f >= n \\
T_b((\exists f. <_n) &=_{def} f < n \\
T_b((\exists f. \leq_n) &=_{def} f <= n
\end{aligned}
$$

El caso de los axiomas de inclusión estrictos que involucran expresiones con dominios concretos plantea ciertas dificultades a la hora de categorizar el uso de las transposiciones. Por ejemplo, consideremos el siguiente axioma de inclusión de clases:

$$Employee \sqcap \exists salary.\ <_{300} \ \sqsubseteq PoorWorker$$

que expresa que los empleados que ganan menos de $300 mensuales son trabajadores pobres. Utilizando un acercamiento similar al empleado con los axiomas de inclusión rebatibles, este axioma de inclusión sería interpretado como la regla estricta:

$$poorworker(X) \leftarrow employee(X), salary(X, Y), X < 300$$

Al considerar las transposiciones de esta regla tenemos las siguientes:

$$\sim employee(X) \leftarrow \sim poorworker(X), salary(X, Y), X < 300$$
$$\sim salary(X, Y) \leftarrow employee(X), \sim poorworker(X), X < 300$$

Pero, ¿cuál es el sentido de una regla como:

$$\sim(X < 300) \leftarrow employee(X), salary(X, Y), \sim poorworker(X),$$

o, en el mejor de los casos,

$$X >= 300 \leftarrow employee(X), salary(X, Y), \sim poorworker(X)?$$

Si bien esta última regla funciona perfectamente en el contexto del intérprete DeLP, tiene el problema que cuando se consulta por un X mayor o igual que 300, en lugar de contestar directamente usando cálculos aritméticos, trata de fabricar un argumento utilizando la regla anterior.

Por ello, en el contexto de esta tesis limitaremos el uso de los axiomas que involucren dominios concretos a las terminologías rebatibles o Dboxes. A continuación, mostraremos cómo el problema de la especificación de criterios de préstamos en un banco puede ser especificado utilizando δ-ontologías enriquecidas con predicados para manipular dominios concretos como los números enteros. Más adelante (veáse la Sección 6.6.4), mostraremos también cómo integrar los δ-formularios con los lenguajes estándar de la Web Semántica a través de la especificación de atributos rebatibles por medio de δ-ontologías.

6.6.3. Caso de estudio: Los préstamos en un banco revisados

Hemos mostrado cómo se pueden extender los formularios web utilizando a la Programación en Lógica Rebatible como lenguaje de representación de conocimiento para modelar el significado de los atributos rebatibles asociados a los campos de los formularios. Sin embargo, creemos que, en el contexto de la Web Semántica, sería más apropiado

utilizar ontologías representadas con un lenguaje con semántica basada en Lógicas para la Descripción para respetar el estándar de la iniciativa de la Web Semántica de la mejor manera posible, y, al mismo tiempo, ser capaces de tener un enfoque automático para lidiar con ontologías que modelan información incompleta y potencialmente inconsistente.

Por ello, a continuación, vamos a presentar una versión revisada del ejemplo en el dominio bancario referente a la obtención de préstamos para ilustrar nuestra propuesta. La presentación se realizará considerando un sistema de integración de ontologías con política *global-as-view* de la misma manera en la que fue definido en la Sección 4.6. Igual que antes, un banco internacional mantiene un seguimiento de sus clientes para determinar si otorga préstamos. Para cada cliente, el banco mantiene el nombre, país de origen, profesión, ingreso promedio mensual y el estado familiar del cliente. El gerente de cuentas del banco posee un número de criterios para otorgar préstamos. Los préstamos son otorgados si la persona tiene un "perfil" razonable, de acuerdo a un conjunto de criterios para el otorgamiento de créditos definidos por el banco. Definiremos una ontología global con criterios para la obtención de préstamos. Además, definiremos ontologías locales para la calificación de riesgo de los distintos países, para los datos de los clientes, y para la información sobre clientes almacenada en el banco.

Ejemplo 6.6.1 *Consideremos una ontología* $\Sigma_{criteria} = (T_S^{criteria}, T_D^{criteria}, \emptyset)$ *que define criterios para el otorgarmiento de créditos en un banco como la presentada en la Figura 6.9. La sentencia (1) de la terminología estricta* $T_S^{criteria}$ *expresa que todos los millonarios son candidatos a préstamos. La terminología rebatible* $T_D^{criteria}$ *está formada por las sentencias (2) a la (9). La sentencia (2) indica que usualmente quien tiene un perfil adecuado de acuerdo al banco es candidato a recibir un préstamo. La sentencia (3) establece que un cliente con altos ingresos (superiores o iguales a* \$1000*) y que pidió un préstamo razonable (inferior a* \$2000*) es usualmente candidato a recibir un préstamo. La sentencia (4) expresa que los clientes con un buen ingreso y provenientes de un país creíble tipicamente tienen un perfil adecuado para el banco. La sentencia (5) dice que los clientes con un ingreso superior a los 300 dólares usualmente son considerados con un buen ingreso. La sentencia (6) establece que las personas insolventes normalmente no tienen un buen ingreso. La sentencia (7) expresa que un cliente cuya profesión es la de estudiante universitario usualmente es insolvente, a menos (como dice la sentencia (8)) que provenga de una familia adinerada. La sentencia (9) dice que si para un cliente se conoce que su registro familiar indica que es rico entonces se considera que proviene de una familia adinerada.*

La δ-ontología $\Sigma_{HSO} = (\emptyset, T_D^{HSO}, A^{HSO})$ *provee información acerca del riesgo asociado a países y se presenta en la Figura 6.10. La información definida en la Dbox* T_D^{HSO} *dice que usualmente los países democráticos son creíbles (sentencia (1)) a menos que se hallen en guerra (sentencia (2)), y que los países democráticos pero con gobiernos corruptos no*

son creíbles (sentencia (3)). La caja asercional A^{HSO} establece que Grecia y Krakosia son países (aserciones (4) y (5)), Grecia es una democracia (aserción (6)), y Krakosia es una democracia en guerra (aserciones (7) y (8)).

Terminología estricta (Sbox) $T_S^{criteria}$:

(1) $millionaire \sqsubseteq candidate$

Terminología rebatible (Dbox) $T_D^{criteria}$:

(2) $profile\_ok \sqsubseteq candidate$
(3) $\exists cstmr\_income. \geq_{1000} \sqcap \exists req\_loan. <_{2000} \sqsubseteq candidate$
(4) $good\_income \sqcap \exists cstmr\_country.credible \sqsubseteq profile\_ok$
(5) $\exists cstmr\_income. >_{300} \sqsubseteq good\_income$
(6) $\neg solvent \sqsubseteq \neg good\_income$
(7) $customer \sqcap \exists cstmr\_profession.univ\_student \sqsubseteq \neg solvent$
(8) $customer \sqcap \exists cstmr\_profession.univ\_student \sqcap rich\_family \sqsubseteq solvent$
(9) $\exists family\_record.rich\_status \sqsubseteq rich\_family$

Figura 6.9: Ontología $\Sigma_{criteria} = (T_S^{criteria}, T_D^{criteria}, \emptyset)$ con un conjunto de criterios para el otorgamiento de créditos a clientes

Además, como mencionamos anteriormente, el banco mantiene información histórica acerca de los clientes del banco de tal manera que cuando algún cliente llena un formulario, el banco es capaz de revisar su historial para aprovechar esta información para lograr una decisión más informada acerca de la viabilidad del otorgamiento de un préstamo.

Ejemplo 6.6.2 *Consideremos la δ-ontología $\Sigma_{bank} = (\emptyset, T_D^{bank}, A^{bank})$ con información de clientes provista por el banco y presentada en la Figura 6.11. La Dbox T_D^{bank} expresa que médico, profesor, abogado, ingeniero, estudiante de maestría, doctorado o de grado y ninguna son profesiones (sentencia (1)); rico, quebrado y ninguno son estados de la familia de un cliente (sentencia (2)); rico es el estado de familia rica (sentencia (3)); los estudiantes de maestría, doctorado y grado son estudiantes universitarios (sentencia (4)), y, la sentencia (5) dice que ser estudiante universitario es un tipo de profesión. Las sentencias (6) y (7) expresan que el atributo family_record tiene como dominio al cliente y como codominio al estado de una familia, respectivamente; las sentencias (8) y (9) establecen que el atributo cstmr_name tiene como dominio al cliente y como codominio al conjunto de nombres, respectivamente; las sentencias (10) y (11) dicen que el atributo profesión tiene como dominio al cliente y como codominio al conjunto de profesiones, respectivamente; las*

Terminología rebatible T_D^{HSO}:

(1) $country \sqcap democracy \sqsubseteq credible$
(2) $country \sqcap democracy \sqcap atwar \sqsubseteq \neg credible$
(3) $country \sqcap democracy \sqcap corruptgovt \sqsubseteq \neg credible$

Caja asercional A^{HSO}:

(4) $krakosia : country$
(5) $greece : country$
(6) $greece : democracy$
(7) $krakosia : democracy$
(8) $krakosia : atwar$

Figura 6.10: Ontología $\Sigma_{HSO} = (\emptyset, T_D^{HSO}, A^{HSO})$ provista por la HSO con información de riesgo sobre países

sentencias (12) y (13) expresan que el atributo país tiene como dominio al cliente y como codominio al conjunto de países, respectivamente; las sentencias (14) y (15) dicen que el atributo ingreso del cliente tiene como dominio al cliente y como codominio al conjunto de números, respectivamente; las sentencias (16) y (17) establecen que el atributo cantidad de dinero pedida en préstamo tiene como dominio al cliente y como codominio al conjunto de los números, respectivamente; finalmente, la sentencia (18) dice que el conjunto de nombres está compuesto por Ajax, Danae, John, Peter y Steve.

La Abox A^{bank} establece que de acuerdo a los registros del banco el cliente c_1 tiene el estado rico (aserción (19)), que el estado de los clientes c_2 y c_3 es desconocido (aserciones (20) y (21)) y, finalmente, la aserción (22) establece que el cliente c_4 es millonario.

Cuando un usuario completa los datos de un formulario se produce una caja asercional temporal, la cual es utilizada para realizar inferencias en el contexto del sistema de integración de ontologías definido más arriba (ver la Definición 6.6.4).

Ejemplo 6.6.3 *En la Figura 6.12, la caja asercional A^{user} dice que hay cinco clientes c_1, c_2, c_3, c_4 y c_5 (aserciones (1)–(5)). El cliente c_1 se llama John, es estudiante de doctorado, proviene de Krakosia, y tiene un ingreso de* \$400 *(aserciones (6)–(9)). El cliente c_2 se llama Ajax, también es estudiante de doctorado, proviene de Grecia, y tiene un ingreso de* \$350 *(aserciones (10)–(13)). El cliente c_3 tiene nombre Danae, no trabaja, es griega y gana* \$1000 *(aserciones (14)–(17)). El cliente c_4 se llama Peter (aserciones (18)). El cliente c_5 sea llama Steve, es un profesor griego que gana* \$1010 *(aserciones (19)–(22)). Finalmente, tenemos información acerca de los montos de préstamo que requiere cada*

Dbox T_D^{bank}:

(1) $\{medical\_doctor, professor, lawyer, engineer, msc\_student, phd\_student, undergrad\_student, none\} \sqsubseteq profession$
(2) $\{rich, broke, unknown\} \sqsubseteq family\_status$
(3) $\{rich\} \sqsubseteq rich\_status$
(4) $\{msc\_student, phd\_student, undergrad\_student\} \sqsubseteq univ\_student$
(5) $univ\_student \sqsubseteq profession$
(6) $\top \sqsubseteq \forall family\_record.family\_status$
(7) $\top \sqsubseteq \forall family\_record^{-}.customer$
(8) $\top \sqsubseteq \forall cstmr\_name.name$
(9) $\top \sqsubseteq \forall cstmr\_name^{-}.customer$
(10) $\top \sqsubseteq \forall cstmr\_profession.profession$
(11) $\top \sqsubseteq \forall cstmr\_profession^{-}.customer$
(12) $\top \sqsubseteq \forall cstmr\_country.country$
(13) $\top \sqsubseteq \forall cstmr\_country^{-}.customer$
(14) $\top \sqsubseteq \forall cstmr\_income.\mathbb{N}$
(15) $\top \sqsubseteq \forall cstmr\_income^{-}.customer$
(16) $\top \sqsubseteq \forall req\_loan.\mathbb{N}$
(17) $\top \sqsubseteq \forall req\_loan^{-}.customer$
(18) $\{ajax, danae, john, peter, steve\} \sqsubseteq name$

Abox A^{bank}:

(19) $\langle c_1, rich\rangle : family\_record$
(20) $\langle c_2, unknown\rangle : family\_record$
(21) $\langle c_3, unknown\rangle : family\_record$
(22) $c_4 : millionaire$

Figura 6.11: Ontología $\Sigma_{bank} = (\emptyset, T_D^{bank}, A^{bank})$ con información de clientes provista por el banco

cliente en particular: John pidió \$2000*; Ajax,* \$4500*; Danae y Peter,* \$1000*, y, Steve,* \$1800*.*

Finalmente, en el contexto de un sistema de integración de ontologías *global-as-view* como el presentado en la Sección 4.6, la ontología $\Sigma_{criteria}$ haría las veces de ontología global mientras que las ontologías Σ_{bank} y Σ_{HSO} conformarían las ontologías fuente. En dicho contexto, necesitamos reglas puente para hacer corresponder los términos de la ontología global con las ontologías locales. En este caso, optamos por la construcción de reglas puente triviales; es decir, por cada concepto de la ontología global $\Sigma_{criteria} : C$ habrá a la fuerza un concepto $\Sigma_{bank} : C$ en la ontología fuente Σ_{bank} o un concepto $\Sigma_{HSO} : C$ en la ontología Σ_{HSO} dependiendo del caso. Entonces, las reglas puente serán siempre de la forma:

$$\Sigma_{HSO} : C \sqsubseteq \Sigma_{criteria} : C, \text{o bien}$$
$$\Sigma_{bank} : C \sqsubseteq \Sigma_{criteria} : C.$$

Ejemplo 6.6.4 *Consideremos la ontología $\Sigma_{criteria}$ y las ontologías Σ_{HSO} y Σ_{bank}. Dos*

Abox A^{user}:

(1)	$c_1 : customer$	(2)	$c_2 : customer$
(3)	$c_3 : customer$	(4)	$c_4 : customer$
(5)	$c_5 : customer$		
(6)	$\langle c_1, john\rangle : cstmr\_name$	(7)	$\langle c_1, phd\_student\rangle : cstmr\_profession$
(8)	$\langle c_1, krakosia\rangle : cstmr\_country$	(9)	$\langle c_1, 400\rangle : cstmr\_income$
(10)	$\langle c_2, ajax\rangle : cstmr\_name$	(11)	$\langle c_2, phd\_student\rangle : cstmr\_profession$
(12)	$\langle c_2, greece\rangle : cstmr\_country$	(13)	$\langle c_2, 350\rangle : cstmr\_income$
(14)	$\langle c_3, danae\rangle : cstmr\_name$	(15)	$\langle c_3, none\rangle : cstmr\_profession$
(16)	$\langle c_3, greece\rangle : cstmr\_country$	(17)	$\langle c_3, 1000\rangle : cstmr\_income$
(18)	$\langle c_4, peter\rangle : cstmr\_name$		
(19)	$\langle c_5, steve\rangle : cstmr\_name$	(20)	$\langle c_5, professor\rangle : cstmr\_profession$
(21)	$\langle c_5, greece\rangle : cstmr\_country$	(22)	$\langle c_5, 1010\rangle : cstmr\_income$
(22)	$\langle c_1, 2000\rangle : req\_loan$	(23)	$\langle c_2, 4500\rangle : req\_loan$
(24)	$\langle c_3, 1000\rangle : req\_loan$	(25)	$\langle c_4, 1000\rangle : req\_loan$
(26)	$\langle c_5, 1800\rangle : req\_loan$		

Figura 6.12: Información asercional provista por el usuario A^{user}

reglas puente posibles son:

$$\begin{aligned} \Sigma_{HSO} : country &\sqsubseteq \Sigma_{criteria} : country, y \\ \Sigma_{bank} : millionaire &\sqsubseteq \Sigma_{criteria} : millionaire. \end{aligned}$$

En la Sección 4.4.2, vimos que en el acercamiento tradicional a la asignación de semántica a las ontologías DL se realiza con semántica basada en teoría de modelos. El problema de tal enfoque reside en que cuando las ontologías involucradas son inconsistentes, como las lógicas DL son un subconjunto de la Lógica de Primer Orden, el mecanismo de inferencia tiene un efecto explosivo ya que cualquier sentencia se convierte en demostrable a partir de la ontología en cuestión. En esta Tesis se propone un acercamiento argumentativo al tratamiento de las inconsistencias en ontologías posiblemente inconsistentes, donde la asignación de semántica a las δ-ontologías se hace por medio de un programa lógico rebatible. Por lo tanto, en el siguiente ejemplo veremos cómo interpretar las ontologías describiendo los atributos rebatibles de un formulario en términos de un programa DeLP que permita determinar sus valores en base a un análisis dialéctico.

Ejemplo 6.6.5 *La interpretación de las ontologías* $\Sigma_{criteria} = (T_S^{criteria}, T_D^{criteria}, \emptyset)$, $\Sigma_{bank} = (\emptyset, T_D^{bank}, A^{bank})$ *y* $\Sigma_{HSO} = (\emptyset, T_D^{HSO}, A^{HSO})$ *se muestran en las Figuras 6.13, 6.14, y 6.16, resp. como los programas DeLP* $\mathcal{P}_{criteria} = (\Pi^{criteria}, \Delta^{criteria})$, $\mathcal{P}_{bank} = (\Pi^{bank}, \Delta^{bank})$, $\mathcal{P}_{HSO} = (\Pi^{HSO}, \Delta^{HSO})$. *La Abox* A^{user} *es interpretada como el conjunto de hechos* Π_{user}

presentado en la Figura 6.17. Cuando consideramos el sistema de integración formado por[8]*:*

$$\Sigma_{merge} = \left(\ T_S^{criteria}, T_D^{criteria} \cup T_D^{bank} \cup T_D^{HSO}, A^{bank} \cup A^{HSO} \cup A^{user} \ \right),$$

el siguiente análisis dialéctico surge a partir la interpretación de Σ_{merge}*. Por ejemplo, al verificar si John es un candidato a un préstamo (*i.e.*, determinar el valor del atributo candidate), hallamos que* c_1 *pertenece potencialmente al concepto candidate ya que existe un argumento* $\langle \mathcal{A}_1, candidate(c_1) \rangle$ *donde:*

$$\mathcal{A}_1 \ = \ \left\{ \begin{array}{l} (candidate(c_1) \prec profile\_ok(c_1)), \\ (profile\_ok(c_1) \prec \\ \qquad good\_income(c_1), cstmr\_country(c_1, krakosia), credible(krakosia)), \\ (credible(krakosia) \prec country(krakosia), democracy(krakosia)), \\ (good\_income(c_1) \prec cstmr\_income(c_1, 400), 400 >= 300) \end{array} \right\}$$

pero este es derrotado por otro argumento $\langle \mathcal{A}_2, \sim credible(krakosia) \rangle$ *que dice que Krakosia no es un país confiable ya que se halla en guerra:*

$$\mathcal{A}_2 \ = \ \left\{ \begin{array}{l} \sim credible(krakosia) \prec \\ \qquad country(krakosia), democracy(krakosia), atwar(krakosia) \end{array} \right\}$$

En el caso de Ajax, el valor del atributo candidate también es indeciso pues no podemos determinar si el individuo c_2 *pertenece en forma justificada al concepto candidate. Existe un argumento* $\langle \mathcal{B}_1, candidate(c_2) \rangle$ *(por lo tanto,* c_2 *pertenece potencialmente al concepto candidate), con:*

$$\mathcal{B}_1 \ = \ \left\{ \begin{array}{l} (candidate(c_2) \prec profile\_ok(c_2)), \\ (profile\_ok(c_2) \prec \\ \qquad good\_income(c_2), cstmr\_country(c_2, greece), credible(greece)), \\ (credible(greece) \prec country(greece), democracy(greece)) \\ (good\_income(c_1) \prec cstmr\_income(c_1, 350), 350 >= 300) \end{array} \right\}$$

pero este argumento es derrotado por otro argumento $\langle \mathcal{B}_2, \sim good\_income(c_2) \rangle$ *con:*

$$\mathcal{B}_2 \ = \ \left\{ \begin{array}{l} \sim good\_income(c_2) \prec \ \sim solvent(c_2) \\ \sim solvent(c_2) \prec customer(c_2), cstmr\_profession(c_2, phd\_student), \\ \qquad univ\_student(phd\_student) \\ univ\_student(phd\_student) \prec phd\_student = phd\_student \end{array} \right\}$$

En el caso de Danae, el atributo candidate toma el valor verdadero pues el individuo c_3 *pertenece en forma justificada al concepto candidate ya que existe un argumento victorioso pues no tiene derrotadores (y, por lo tanto, está garantizado)* $\langle \mathcal{C}_1, candidate(c_3) \rangle$ *donde:*

[8] En un abuso de notación, las reglas puente no se incluyen por simplicidad de la presentación.

$$\mathcal{B}_1 \;=\; \left\{ \begin{array}{l} (candidate(c_3) \prec profile\_ok(c_3)), \\ (profile\_ok(c_3) \prec \\ \qquad good\_income(c_3), cstmr\_country(c_3, greece), credible(greece)), \\ (credible(greece) \prec country(greece), democracy(greece)), \\ (good\_income(c_3) \prec cstmr\_income(c_3, 1000), 1000 >= 300) \end{array} \right\}$$

Por otro lado, Peter también es un candidato a recibir un préstamo pues existe un argumento formado sólamente por reglas estrictas que expresa que es millonario; así, el individuo c_4 pertenece estrictamente al concepto candidate. Este argumento como está basado en la regla estricta "$candidate(peter) \leftarrow millionaire(peter)$". De esta manera, el argumento no tiene derrotadores y está trivialmente garantizado.

Finalmente, en el caso de Steve tenemos dos argumentos que nos dicen que Steve es un candidato a préstamo. Por un lado, tenemos un argumento $\langle \mathcal{C}_1, candidate(c_5) \rangle$ expresando que Steve es un candidato a préstamo pues tiene el perfil adecuado, ya que tiene ingreso adecuado y proviene de un país confiable:

$$\mathcal{C}_1 \;=\; \left\{ \begin{array}{l} (candidate(c_5) \prec profile\_ok(c_5)), \\ (profile\_ok(c_5) \prec \\ \qquad good\_income(c_5), cstmr\_country(c_5, greece), credible(greece)), \\ (credible(greece) \prec country(greece), democracy(greece)), \\ (good\_income(c_5) \prec cstmr\_income(c_3, 1010), 1010 >= 300) \end{array} \right\}$$

Sin embargo, también existe otro argumento (sin derrotadores) que expresa que Steve es candidato pues tiene un ingreso superior a \$1000 *y ha pedido un monto inferior a* \$2000*. Formalmente, existe un argumento $\langle \mathcal{C}_2, candidate(c_5) \rangle$ donde:*

$$\mathcal{C}_2 \;=\; \left\{ \begin{array}{l} candidate(c_5) \prec \\ \qquad cstmr\_income(c_5, 1010), 1010 >= 1000, \\ \qquad req\_loan(c_5, 1800), 1800 < 2000 \end{array} \right\}$$

Así, podemos concluir que el individuo c_5 pertenece estrictamente al concepto candidate.

6.6.4. Enlace de δ-formularios con δ-ontologías

En esta sección mostramos cómo integrar la noción de δ-formulario definida previamente (en la Sección 6.3.3) con los acercamientos actuales a la Web Semántica. Previamente, vimos cómo extender los formularios web tradicionales para incorporar atributos rebatibles expresados en términos de un programa DeLP, caracterizando la noción de *formulario con atributos rebatibles* o δ-formulario. En esta sección, nuestro objetivo es brindar una manera de asociar una δ-ontología Σ con un esquema de formulario arbitrario (el cual,

Reglas estrictas $\Pi^{criteria}$:

(1) $candidate(X) \leftarrow millionaire(X)$.
(1') $\sim millionaire(X) \leftarrow \sim candidate(X)$.

Reglas rebatibles $\Delta_D^{criteria}$:

(2) $candidate(X) \prec profile\_ok(X)$.
(3) $candidate(X) \prec cstmr\_income(X, Y), Y >= 1000,$
$req\_loan(X, Z), Z < 2000$.
(4) $profile\_ok(X) \prec good\_income(X), cstmr\_country(X, Y), credible(Y)$.
(5) $good\_income(X) \prec cstmr\_income(X, Y), Y > 300$.
(6) $\sim good\_income(X) \prec \sim solvent(X)$.
(7) $\sim solvent(X) \prec customer(X), cstmr\_profession(X, Y), univ\_student(Y)$.
(8) $solvent(X) \prec customer(X), cstmr\_profession(X, Y),$
$univ\_student(Y), rich\_family(X)$.
(9) $rich\_family(X) \prec family\_record(X, Y), rich\_status(Y)$.

Figura 6.13: Ontología $\Sigma_{criteria} = (T_S^{criteria}, T_D^{criteria}, \emptyset)$ con un conjunto de criterios para el otorgamiento de créditos a clientes expresada como un programa DeLP $(\Pi^{criteria}, \Delta^{criteria})$

igual que antes, va a corresponder a un número de posibles instancias de formulario diferentes). Se asumirá que la δ-ontología Σ representa conocimiento declarativo asociado con el dominio del problema del esquema de formulario. Así, como se discutió en los ejemplos previos, un esquema de formulario correspondiendo a una aplicación bancaria podría contener una ontología asociada que represente políticas tentativas (y posiblemente conflictivas) para el otorgamiento de créditos. Por lo tanto, a continuación redefinimos las nociones de esquema de formulario, instancia de formulario, hechos de formulario, esquema de formulario con atributos rebatibles e instancia de δ-formulario para enlazarlos a la noción de δ-ontología para hacerlos de esta manera viables para su implementación en el contexto de la iniciativa de la Web Semántica.

Definición 6.6.3 (Esquema de formulario. Instancia de formulario (revisadas). *Un esquema de formulario es una n-upla $\mathcal{F} = \langle F, T, C, c_j \rangle$, donde $F = [f_1, \ldots, f_n]$ es una lista de campos de formulario, $T = [T_1, \ldots, T_n]$ es una lista de tipos o conceptos, C es el concepto al que pertenece el individuo del mundo externo que llena los campos del formulario, y c_j es el identificador del individuo que llena los campos del formulario. Si $V = [v_1, \ldots, v_n]$ es una lista de valores tal que $v_i \in T_i, i = 1, \ldots, n$ es el valor asociado para $f_i \in F$ para el individuo c_j. Una instancia de formulario basada en $\mathcal{F}$ con valor V e identificador de individuo c_j (denotado como $\mathcal{F}_V^{c_j}$) es una tripla $\mathcal{F}_{c_j}^V = \langle F, V, c_j \rangle$.*

Reglas rebatibles Δ^{bank}:

(1.a) $profession(X) \prec X = medical\_doctor.$
(1.b) $profession(X) \prec X = professor.$
(1.c) $profession(X) \prec X = lawyer.$
(1.d) $profession(X) \prec X = engineer.$
(1.e) $profession(X) \prec X = msc\_student.$
(1.f) $profession(X) \prec X = phd\_student.$
(1.g) $profession(X) \prec X = undergrad\_student.$
(1.h) $profession(X) \prec X = none.$
(2.a) $family\_status(X) \prec X = rich.$
(2.b) $family\_status(X) \prec X = broke.$
(2.c) $family\_status(X) \prec X = unknown.$
(3) $rich\_status(X) \prec X = rich.$
(4.a) $univ\_student(X) \prec X = msc\_student.$
(4.b) $univ\_student(X) \prec X = phd\_student.$
(4.c) $univ\_student(X) \prec X = undergrad\_student.$

Figura 6.14: Ontología $\Sigma_{bank} = (\emptyset, T_D^{bank}, A^{bank})$ con información de clientes provista por el banco expresada como el programa DeLP $\mathcal{P}_{bank} = (\Pi^{bank}, \Delta^{bank})$ (primera parte)

Ejemplo 6.6.6 *Sean F una lista de campos y T una lista de valores tales que:*

$$F = [cstmr\_name, cstmr\_profession, cstmr\_income, req\_loan, cstmr\_country]$$
$$T = [name, profession, \mathbb{N}, \mathbb{N}, country],$$

entonces $\mathcal{F} = \langle F, T, customer, c_1\rangle$ es un esquema de formulario. Sea V una lista de valores tal que:

$$V = [john, phd\_student, 400, 2000, krakosia],$$

entonces $\mathcal{F}_{c_1}^V = \langle F, V, c_1\rangle$ es una instancia de formulario basada en $\mathcal{F}$.

Dado un esquema de formulario $\mathcal{F} = \langle F, T, C, c_j\rangle$ y una instancia de formulario $\mathcal{F}_V^{c_j}$, capturaremos el conocimiento factual involucrado en $\mathcal{F}_V$ en términos de una Abox $A_{\mathcal{F}_V^{c_j}}$. Tales aserciones serán obtenidas introduciendo nuevos nombres de conceptos asociados con aquellos nombres de campos en un formulario $\mathcal{F}$ y nuevos nombres de constantes correspondientes a los valores presentes en V. Formalmente:

Definición 6.6.4 (Caja asercional de formulario $A^{\mathcal{F}_V}$) *Sea $\mathcal{F} = \langle F, T, C, c_j\rangle$ un esquema de formulario, con $F = [f_1, \ldots, f_n]$, y sea $\mathcal{F}_V^{c_j}$ una instancia de formulario. Definimos la caja asercional de formulario $A^{\mathcal{F}_V^{c_j}}$ como el conjunto:*

$$A_{\mathcal{F}_V^{cj}} =_{def} \{\ C(c_j), f_1(c_j, v_1), \ldots, f_n(c_j, v_n)\ \}.$$

Reglas rebatibles Δ^{bank} (continuación):

(5) $profession(X) \prec univ\_student(X)$.
(6) $family\_status(Y) \prec family\_record(X, Y)$.
(7) $customer(X) \prec family\_record(X, Y)$.
(8) $name(Y) \prec cstmr\_name(X, Y)$.
(9) $customer(X) \prec cstmr\_name(X, Y)$.
(10) $profession(Y) \prec cstmr\_profession(X, Y)$.
(11) $customer(X) \prec cstmr\_profession(X, Y)$.
(12) $country(Y) \prec cstmr\_country(X, Y)$.
(13) $customer(X) \prec cstmr\_country(X, Y)$.
(14) $number(Y) \prec cstmr\_income(X, Y)$.
(15) $customer(X) \prec cstmr\_income(X, Y)$.
(16) $number(Y) \prec req\_loan(X, Y)$.
(17) $customer(X) \prec req\_loan(X, Y)$.
(18.a) $name(X) \prec X = ajax$.
(18.b) $name(X) \prec X = danae$.
(18.c) $name(X) \prec X = john$.
(18.d) $name(X) \prec X = peter$.
(18.e) $name(X) \prec X = steve$.

Hechos Π^{bank}:

(19) $family\_record(c_1, rich)$
(20) $family\_record(c_2, unknown)$
(21) $family\_record(c_3, unknown)$
(22) $millionaire(c_4)$

Figura 6.15: Ontología $\Sigma_{bank} = (\emptyset, T_D^{bank}, A^{bank})$ con información de clientes provista por el banco expresada como el programa DeLP $\mathcal{P}_{bank} = (\Pi^{bank}, \Delta^{bank})$ (continuación)

Ejemplo 6.6.7 (Continúa el Ejemplo 6.6.6) *Dada la instancia de formulario presentada en el Ejemplo 6.6.6, la caja asercional del formulario correspondiente es el conjunto:*

$$A_{\mathcal{F}_V^{c1}} = \left\{ \begin{array}{l} customer(c_1), cstmr\_name(c_1, john), \\ cstmr\_profession(c_1, phd\_student), cstmr\_income(c_1, 400), \\ req\_loan(c_1, 2000), cstmr\_country(c_1, krakosia) \end{array} \right\}.$$

A continuación, mostraremos cómo los valores de campos pueden ser integrados con una δ-ontología arbitraria $\Sigma = (T_S, T_D, A)$ adaptando así el concepto de δ-formulario presentado previamente en la Definición 6.3.3 pero ahora adaptado a un marco DL. Formalmente:

Definición 6.6.5 (Formulario con atributos rebatibles. Instancia de δ-formulario (revisadas)) *Sea $\mathcal{F} = \langle F, T, C, c_j \rangle$ un esquema de formulario, y $\Sigma = (T_S, T_D, A)$ una δ-*

Reglas rebatibles Δ^{HSO}:

(1) $credible(X) \prec country(X), democracy(X)$.
(2) $\sim credible(X) \prec country(X), democracy(X), atwar(X)$.
(3) $\sim credible(X) \prec country(X), democracy(X), corruptgovt(X)$.

Hechos Π^{HSO}:

(4) $country(krakosia)$
(5) $country(greece)$
(6) $democracy(greece)$
(7) $democracy(krakosia)$
(8) $atwar(krakosia)$

Figura 6.16: Ontología $\Sigma_{HSO} = (\emptyset, T_D^{HSO}, A^{HSO})$ provista por la HSO con información sobre países expresada como el programa DeLP $\mathcal{P}_{HSO} = (\Pi^{HSO}, \Delta^{HSO})$

ontología. Un esquema de formulario con atributos rebatibles (o esquema de δ-formulario) $\mathcal{D}$ es un par $\langle \mathcal{F}, \Sigma \rangle$. Si V es un conjunto de valores para el formulario $\mathcal{F}$, una instancia de δ-formulario $\mathcal{D}_V$ es un par $\langle \mathcal{F}_V, \Sigma \rangle$. El conjunto de atributos rebatibles para $\mathcal{D}_V$ se define como el conjunto de predicados $\mathsf{Pred}(\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A) \cup \mathfrak{T}_\Pi(A_{\mathcal{F}_V^{c_j}}), \mathfrak{T}_\Delta(T_D))$.

Nuevamente, dado un esquema de δ-formulario $\langle \mathcal{F}_V^{cj}, \Sigma \rangle$, la definición anterior tiene como objetivo identificar atributos en el formulario $\mathcal{F}$ codificados por el diseñador del formulario como predicados distinguidos en el programa $\mathcal{P} = (\mathfrak{T}_\Pi(T_S) \cup \mathfrak{T}_\Pi(A) \cup \mathfrak{T}_\Pi(A_{\mathcal{F}_V^{c_j}}), \mathfrak{T}_\Delta(T_D))$. Tales atributos se dirán *rebatibles* porque sus valores asociados serán determinados por consultas DeLP resueltas con respecto al programa $\mathcal{P}$. Así, un cambio en el programa DeLP subyacente resultará en un cambio en el valor de tales atributos. Como se mencionó anteriormente, los atributos rebatibles representarán atributos relevantes para el diseñador del formulario, cutos valores dependerán de la ontología codificando el conocimiento relevante del dominio con la adición de aserciones particulares para representar los valores de campos para una dada instancia de formulario.

Ejemplo 6.6.8 *Si el individuo c_1 llamado John (como se presenta en la Figura 6.12) completa el formulario (como en la Figura 6.4), se obtiene una instancia de formulario como la presentada en el Ejemplo 6.6.6. En el Ejemplo 6.6.5, vimos que al consultar por la pertenencia del individuo llamado "John" al concepto "candidato", la misma no pudo ser determinada ya que no había un argumento garantizado a favor del literal "candidate(john)". Luego, cuando un proceso análisis dialéctico es llevado a cabo sobre el programa DeLP obtenido a partir de $\Sigma_{john} = (T_S^{criteria}, T_D^{criteria} \cup T_D^{bank} \cup T_D^{HSO}, A^{bank} \cup A^{HSO} \cup A_{\mathcal{F}_V^{c_1}})$ la*

Hechos Π^{user}:

(1)	$customer(c_1)$	(2)	$customer(c_2)$
(3)	$customer(c_3)$	(4)	$customer(c_4)$
(5)	$customer(c_5)$		
(6)	$cstmr\_name(c_1, john)$	(7)	$cstmr\_profession(c_1, phd\_student)$
(8)	$cstmr\_country(c_1, krakosia)$	(9)	$cstmr\_income(c_1, 400)$
(10)	$cstmr\_name(c_2, ajax)$	(11)	$cstmr\_profession(c_2, phd\_student)$
(12)	$cstmr\_country(c_2, greece)$	(13)	$cstmr\_income(c_2, 350)$
(14)	$cstmr\_name(c_3, danae)$	(15)	$cstmr\_profession(c_3, none)$
(16)	$cstmr\_country(c_3, greece)$	(17)	$cstmr\_income(c_3, 1000)$
(18)	$cstmr\_name(c_4, peter)$		
(19)	$cstmr\_name(c_5, steve)$	(20)	$cstmr\_profession(c_5, professor)$
(21)	$cstmr\_country(c_5, greece)$	(22)	$cstmr\_income(c_5, 1010)$
(22)	$req\_loan(c_1, 2000)$	(23)	$req\_loan(c_2, 4500)$
(24)	$req\_loan(c_3, 1000)$	(25)	$req\_loan(c_4, 1000)$
(26)	$req\_loan(c_5, 1800)$		

Figura 6.17: Información asercional provista por el usuario A^{user} expresada como un conjunto de hechos

pertenencia del individuo c_1 al concepto candidate no podrá ser determinada con respecto al formulario $\langle \mathcal{F}, \Sigma_{john} \rangle$ y no se le otorgará el préstamo.

Cuando Ajax llena el formulario, la pertenencia del individuo c_2 al concepto se trata de corroborar con respecto a la ontología: $\Sigma_{ajax} = (T_S^{criteria}, T_D^{criteria} \cup T_D^{bank} \cup T_D^{HSO}, A^{bank} \cup A^{HSO} \cup A_{\mathcal{F}_{V_2}^{c^2}})$. Nótese que, en este caso, c_2 corresponde al identificador de Ajax y la lista de valores de campos de formulario $V_2 = [ajax, phd\_student, 350, 4500, greece]$. Entonces, el conjunto $A_{\mathcal{F}_{V_2}^{c^2}}) = \{(2), (10)\text{-}(13), (23)\}$, donde estos elementos son considerados respecto del conjunto A^{user}. Como vimos en el Ejemplo 6.6.5, no es posible garantizar el literal "$candidate(ajax)$". Luego, no se le otorgará un préstamo a Ajax.

Cuando Danae llena el formulario, como la pertenencia del individuo c_3 al concepto candidate será determinada con respecto al formulario $\langle \mathcal{F}, \Sigma_{danae} \rangle$ donde $\Sigma_{danae} = (T_S^{criteria}, T_D^{criteria} \cup T_D^{bank} \cup T_D^{HSO}, A^{bank} \cup A^{HSO} \cup \{(3), (13)\text{-}(16), (20)\})$, a ella sí se le otorgará un préstamo.

En el caso de Peter y Steve, como vimos también en el Ejemplo 6.6.5, que existen literales garantizados "$candidate(peter)$" y "$candidate(steve)$", a ambos les será otorgado un préstamo.

6.7. Trabajo relacionado

Según el análisis del estado del arte llevado a cabo, no hay otros trabajos en el área para la introducción de conocimiento rebatible en formularios web como se ha hecho en esta tesis. Investigaciones recientes (Wu et al., 2004) se han centrado en desarrollar una metodología para diseñar sistemas de soporte a la decisión basados en formularios, el cual utiliza factorización y síntesis para procesar conocimiento involucrado en formularios. El marco resultante permite la creación y modificación flexible de formularios generados por computadora útiles para la toma de decisiones y ideales para simplificar el proceso de generación de reportes. Sin embargo, a pesar de que este acercamiento explota la semántica del conocimiento implícito en los formularios, no provee ninguna conexión con sistemas basados en web ni con el manejo de conocimiento rebatible.

(Dong et al., 2004) enfatizan la adopción de la tecnología XML en el desarrollo de sistemas de soporte a la toma de decisiones basados en Web (en inglés, *web-enabled*) eficientes y flexibles. Basado en un caso de estudio para sistema de selección de carteras de inversión (en inglés, *portfolio*), exploran las aristas de diseño en la aplicación de XML para superar el problema de la heterogeneidad del intercambio de datos y el mantenimiento de varios modelos de optimización de portfolios. Sin embargo, en contraste con nuestro acercamiento, (Dong et al., 2004) no manejan situaciones conflictivas como las presentadas aquí, ni integran la argumentación como parte de las estrategias para la toma de decisiones.

Un trabajo pionero en el área de representación de ontologías para la Web Semántica es SHOE (Heflin et al., 2003). Como en nuestro acercamiento, SHOE combina características de los lenguajes de marcado, la representación de conocimiento y las ontologías. Su estructura básica consiste de ontologías, las cuales definen reglas que guían qué tipo de aserciones pueden ser realizadas y qué tipo de inferencias pueden ser obtenidas a partir de afirmaciones fijas (en inglés, *ground assertions*), e instancias (de reglas) que hacen aserciones basadas en dichas reglas. A diferencia de nuestro acercamiento, SHOE sólo implementa reglas Horn estrictas mientras que nosotros brindamos la capacidad de poder representar reglas rebatibles. Además, SHOE ha sido diseñado para eliminar la posiblidad de la ocurrencia de contradicciones ya que no permite la negación lógica. En cambio, nuestro acercamiento no solamente permite el uso de las negaciones sino que maneja las contradicciones lógicas por medio de un análisis dialéctico, haciéndolo apropiado para la interacción entre agentes en el marco de apliaciones de la Web Semántica.

En dirección similar a nuestro trabajo, el razonamiento rebatible basado en reglas en el contexto de la Web Semántica ha motivado el desarrollo de sistemas alternativos como DR-DEVICE (Bassiliades et al., 2004), el cual es capaz de razonar acerca de metadatos RDF sobre múltiples recursos Web utilizando reglas en lógica rebatible (Antoniou et al., 2001, 2004*b*). En contraste con nuestro enfoque, este sistema es implementado sobre un sistema de producción de reglas CLIPS, mientras que nuestra propuesta depende de

la computación de la garantía realizada por el motor de inferencias DeLP utilizando técnicas de la programación en lógica. Además, el criterio de comparación de reglas en la lógica rebatible (de (Antoniou et al., 2001)) es realizado sobre la base de una relación de superioridad mientras que nuestro acercamiento utiliza un criterio de comparación modular entre argumento (el cual, por esta razón puede ser modificado para adaptarlo a las necesidades del dominio de aplicación considerado). Además, DeLP no necesita la adición de *reglas derrotadoras* porque el sistema automáticamente encontrará todos los contraargumentos posibles sobre la base de los argumentos que es capaz de construir, y decidirá sobre la relación de derrota utilizando el criterio de comparación de argumentos provisto. Así, un programa DeLP no necesita codificar excepciones en forma explícita.

La Iniciativa para el Marcado de Reglas (en inglés, *Rule Markup Initiative*) constituye un esfuerzo conjunto hacia la definición de un lenguaje de marcado de reglas llamado *Rule Markup Language* (RuleML) (Boley et al., 2004; RuleML, 2005), permitiendo encadenamiento de reglas XML hacia adelante o hacia atrás para realizar deducción, reescritura, y otras tareas de inferencias y transformaciones. Nuestro acercamiento para codificar programas DeLP comparte en parte alguno de los objetivos de la iniciativa RuleML. Sin embargo, al momento de las publicaciones en las que está basado este capítulo de la Tesis, el lenguaje RuleML no provee marcadores para integrar implicaciones rebatibles, argumentos, y árboles de argumentación, como son presentados aquí.

Antoniou *et al.* (Antoniou et al., 2004*a*) reportan la construcción de un sistema de razonamiento rebatible que puede llegar a ser utilizado como la capa de razonamiento de la Web Semántica. Reportan también que parte su trabajo ha sido la creación de un DTD para traducir teorías rebatibles en archivos XML. De hecho, su DTD es una extensión del DTD de la iniciativa *RuleML* (RuleML, 2005), que cubre reglas estrictas y rebatibles así como la relación de superioridad para reglas definida en su sistema de razonamiento rebatible.

Como se discutió en la Sección 6.4, X-DeLP ofrece un lenguaje de representación de conocimiento que puede ser usado para caracterizar ontologías, usando el motor de inferencias argumentativo subyacente para realizar inferencias. En este respecto, el uso de la argumentación para modelar ingeniería ontológica ha sido recientemente el foco de investigación desde una perspectiva social, *i.e.*se ve a la definición de una ontología compartida como un proceso social. Durante la discusión, los participantes intercambian argumentos los cuales pueden agregar soporte a u objetar ciertas decisiones de ingeniería ontológica. Tempich *et al.*. (Tempich et al., 2005) presentan una ontología que formaliza los principales conceptos las cuales son usadas en una discusión de ingeniería ontológica y así permite trazar argumentos y permite la detección de inconsistencia. Su acercamiento, sin embargo, difiere del nuestro en el sentido que el foco está en la provisión de una ontología para modelar argumentación, donde nuestra propuesta se enfoca en utilizar el lenguaje X-DeLP como un vehículo para modelar ontologías en general y utilizar el motor

de inferencia argumentativo como un mecanismo de razonamiento.

OWL Rules Language es una primera propuesta para extender el lenguaje OWL con reglas con forma de cláusulas de Horn (Horrocks and Patel-Schneider, 2003); incluye un sintaxis abstracta de alto nivel para reglas Horn en OWL DL y OWL Lite. En la misma dirección que nuestra propuesta permite extender OWL para representar reglas de Horn pero no permite la representación de reglas rebatibles y estrictas, ni mucho menos de argumentos y árboles dialécticos.

RIF-Core (el Núcleo del Formato de Intercambio de Información, o, en inglés, *Core of the Rule Interchange Format*) (Boley et al., 2008), desde una perspectiva teórica, corresponde al lenguaje de las reglas Horn definidas sin símbolos de función (o Datalog) y una semántica estándar de primer orden. Al mismo tiempo RIF-Core es un lenguaje de reglas de producción permitiendo solamente especificar acciones. Sintácticamente, RIF-Core tiene un número de extensiones Datalog para soportar características tales como objetos y marcos como en *F-Logic*, identificadores de recursos internacionalizados (IRIs) como identificadores para conceptos, y tipos de datos XML-Schema. Además, *RIF and OWL Compatibility* (RIF-RDF+OWL) define la sintaxis y semántica de la integración de los lenguajes RIF, RDF y OWL. Por lo tanto, RIF está diseñado para permitir la interoperabilidad entre lenguajes de reglas en general y su uso no está limitado a la Web. La novedad de la propuesta de RIF nos hace pensar en extensiones futuras para X-DeLP; en particular, se podría estudiar cómo el lenguaje X-DeLP podría ser adaptado a la RIF, o, por el contrario, cómo extender la propuesta de RIF para representar rebatibilidad de reglas.

6.8. Resumen

En este capítulo hemos presentado un acercamiento novedoso para enriquecer los formularios tradicionales para ambientes basados en aplicaciones web, los cuales pueden adaptarse convenientemente para tecnologías existentes de lenguajes de marcado como XHTML. Nuestra propuesta involucra la posibilidad de modelar inferencias basadas en conceptos que son parte del significado pretendido de un formulario, el cual se puede modelar a través de atributos rebatibles.

Hemos mostrado que el uso de un intérprete DeLP embebido en el lado del cliente permite al diseñador de formularios desarrollar esquemas de formularios más ricos, en los cuales la interacción de atributos rebatibles es tomada en cuenta como parte de la "conducta" del formulario. Las bases de conocimiento de los formularios son expresadas en una manera declarativa, haciendo más fácil enriquecer un formulario, por ejemplo, combinando dos bases de conocimiento existentes. Debe ser notado que nuestro acercamiento asume que toda la información disponible es codificada del lado del cliente (aplicación *browser*), lo cual genera un número de problemas de seguridad y privacidad. En el caso

de la aplicación bancaria, el banco probablemente desearía mantener sus criterios ocultos, de tal manera que los mismos no pudieran ser utilizados con fines maliciosos por el cliente o por terceras partes. En tales casos, consideraciones de seguridad del lado del cliente necesitarían ser aplicadas (*e.g.*, la incorporación de técnicas de encriptación), las cuales son corrientemente un aspecto bajo investigación. Alternativamente, un acercamiento similar puede ser llevado a cabo en el lado del servidor o utilizando tecnologías tales como AJAX (*Asynchronous Java And XML*) (Asleson and Schutta, 2005), realizadas en parte sobre el cliente y en parte sobre el servidor, en las cuales la división exacta de tareas entre el cliente y el servidor son decididas en tiempo de ejecución (posiblemente en una base caso por caso). La realización de tal división dinámica puede requerir la aplicación de un acercamiento "distribuido" a la argumentación, como se discute en (Brena et al., 2006).

La implementación de la redefinición de programas es implementable sencillamente y ofrece una atractiva posibilidad para integrar bases de conocimiento rebatibles de diferentes fuentes (como $\mathcal{P}_{bank}$ y $\mathcal{P}_{sec}$). Claramente, consideraciones ontológicas adicionales (por ej., asunción de nombres únicos[9]) son requeridas para tales operaciones de combinación; extender tales consideraciones es parte del trabajo futuro.

[9]En inglés, *unique name assumption* o UNA (Brewka et al., 1997).

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

La World Wide Web actual está basada principalmente en documentos escritos para su presentación visual para usuarios humanos. Sin embargo, para obtener todo el potencial de la web es necesario que los programas de computadoras o agentes sean capaces de comprender la información allí presente. En este sentido, La Web Semántica es una visión futura de la web donde la información tiene significado exacto, permitiendo así que las computadoras entiendan y razonen en base a la información hallada en la Web. En particular, la Web Semántica propone resolver el problema de la asignación de semántica a los recursos web por medio de metadatos cuyo significado es dado a través de definiciones de ontologías.

Tal como se detalló en el Capítulo 1, la motivación de la Web Semántica es construir un espacio de representación de conocimiento que permita a los agentes inteligentes con poder delegado por sus usuarios ser capaces de comprender la información allí presente y, de esta manera, poder tomar decisiones a favor de dichos usuarios. El significado de esta información es definido por medio de ontologías, que son formalizaciones del conocimiento de un dominio de aplicación. Sin embargo, a pesar de que las definiciones de ontologías expresadas en Lógicas para la Descripción pueden ser procesadas por razonadores estándar, tales razonadores son incapaces de lidiar con ontologías inconsistentes.

Los sistemas argumentativos constituyen una formalización del razonamiento rebatible donde se pone especial énfasis en la noción de argumento. En este contexto, la construcción de argumentos permite que un agente obtenga conclusiones en presencia de información incompleta y potencialmente contradictoria. En esta Disertación, hemos propuesto un marco formal para el razonamiento con ontologías incompletas y potencialmente inconsistentes en el marco de un sistema argumentativo como lo es la Programación en Lógica Rebatible, la cual se vio en detalle en el Capítulo 3.

La importancia estratégica de la definición de ontologías para poder llevar a cabo

la realización de la iniciativa de la Web Semántica junto con la presencia de ontologías incompletas y potencialmente contradictorias motivó el desarrollo del marco de razonamiento con δ-ontologías presentado en el Capítulo 4. Investigaciones previas de otros autores, determinaron que un subconjunto de las Lógicas para la Descripción pueden ser traducidas efectivamente a un conjunto de la Programación en Lógica. Entonces, nuestra propuesta involucra corresponder ontologías expresadas en Lógicas para la Descripción con Programas Lógicos Rebatibles, para lidiar con definiciones de ontologías inconsistentes en la Web Semántica. Esto es, dada una ontología Σ_{OWL} expresada en el lenguaje OWL-DL, es posible construir una ontología Σ_{DL} equivalente expresada en las Lógicas para la Descripción. En el caso en que Σ_{DL} satisfaga ciertas restricciones, ésta puede ser expresada como un programa DeLP $\mathcal{P}_{\Sigma}$. Por lo tanto, dada una consulta ϕ acerca de la pertenencia de una instancia a a un cierto concepto C expresada con respecto a Σ_{OWL}, se realiza un análisis dialéctico para determinar todas las razones a favor y en contra de la plausibilidad de la afirmación $C(a)$.

Como también se mostró en el Capítulo 4, la integración de datos es el problema de combinar datos residiendo en diferentes fuentes y el de proveer al usuario con una vista unificada de dichos datos. El problema de diseñar sistemas de integración de datos es particularmente importante en el contexto de aplicaciones en la Web Semántica donde las ontologías son desarrolladas independientemente unas de otras, y por esta razón pueden ser mutuamente inconsistentes. Dada una ontología, nos interesa conocer en qué condiciones un individuo es una instancia de un cierto concepto. Como cuando se tienen varias ontologías, los mismos conceptos pueden tener nombres distintos para un mismo significado o aún nombres iguales para significados diferentes, para relacionar los conceptos entre dos ontologías diferentes se utilizaron reglas puente o reglas de articulación. De esta manera, un concepto se corresponde a una vista sobre otros conceptos de otra ontología. Se mostró también bajo qué condiciones la propuesta del razonamiento con δ-ontologías puede ser adaptado a los dos tipos de integración de ontologías *global-as-view* y *local-as-view* considerados en la literatura especializada.

El Capítulo 5 se focalizó en el estudio de las propiedades formales de la propuesta de razonamiento con δ-ontologías. En particular, para la evaluación se recurrió al marco propuesto por Huang et al. (2004). Así, analizamos las propiedades formales que se desprenden de este acercamiento novedoso al tratamiento de ontologías inconsistentes en la Web Semántica. Los principales resultados obtenidos fueron que, como la interpretación de δ-ontologías como Programas Lógicos Rebatibles es realizada a través de una función de transformación que preserva la semántica de las ontologías involucradas, los resultados obtenidos al realizar consultas Aboxes son sensatos. Esto quiere decir que, cuando las ontologías consideradas son consistentes, las respuestas obtenidas utilizando la Programación en Lógica Rebatible como substrato semántico de las δ-ontologías produce resultados que coinciden con aquellos proporcionados por los mecanismos de razonamiento tradicio-

nales de las Lógicas para la Descripción. Además, mostramos que el operador presentado es además consistente y significativo. Esto es, que el mismo no produce respuestas contradictorias en presencia de ontologías inconsistentes y, por lo tanto, sirve para extender el mecanismo de razonamiento de las ontologías tradicionales en los casos en que los mecanismos de razonamiento estándar no son capaces de brindar una respuesta. Otros resultados mostrados están relacionados con una notación para extender la notación UML llamada δ-UML, la que permite caracterizar gráficamente algunos de los patrones de razonamiento que aparecen al representar conocimiento con δ-ontologías.

Además, hemos mostrado cómo la propuesta presentada puede ser extendida para implementar un operador de mezcla de ontologías. Este operador está basado en resultados tomados prestados del área de Revisión de Creencias. Así, mostramos cómo puede gestionarse la unión de la información terminológica estricta (o Sboxes) de las ontologías involucradas cuando la misma produce inconsistencia. El efecto final está dado porque parte de la información estricta de una ontología inconsistente, es transformada en información ontológica tentativa.

En virtud de los avances realizados en los Capítulos 4 y 5, vimos que dicho acercamiento al razonamiento en presencia de ontologías inconsistentes brinda la posibilidad de abordar de una manera eficaz ciertos problemas de aplicación del ámbito del comercio electrónico, donde el modelo de reglas de negocio puede ser especificado en términos de ontologías. Entonces, la capacidad de razonar frente a ontologías inconsistentes permite abordajes alternativos conceptualmente más claros, ya que es posible automatizar ciertas decisiones de negocios tomadas a la luz de un conjunto de reglas de negocio posiblemente inconsistentes expresadas como una o varias ontologías y, al mismo tiempo, tener un sistema capaz de brindar una explicación del porqué se arribó a una conclusión determinada.

En consecuencia, en el Capítulo 6 presentamos una aplicación del razonamiento sobre ontologías inconsistentes por medio de la argumentación rebatible al modelado de formularios en la World Wide Web. La noción de los formularios como una manera de organizar y presentar datos ha sido utilizada desde el comienzo de la World Wide Web. Los formularios Web han evolucionado junto con el desarrollo de nuevos lenguajes de marcado, en los cuales es posible proveer guiones de validación como parte del código del formulario para verificar si el significado pretendido del formulario es correcto. Sin embargo, para el diseñador del formulario, parte de este significado pretendido frecuentemente involucra otras características que no son restricciones por sí mismas, sino más bien atributos emergentes del formulario, los cuales proveen conclusiones plausibles en el contexto de información incompleta y potencialmente contradictoria. Como el valor de tales atributos puede cambiar en presencia de nuevo conocimiento, los llamamos atributos rebatibles.

Propusimos entonces extender los formularios Web para incorporar atributos rebatibles como parte del conocimiento que puede ser codificado por el diseñador del formulario. Vimos cómo dicho conocimiento puede ser especificado mediante un programa DeLP, y

posteriormente, como una ontología expresada en Lógicas para la Descripción. Así, la extensión propuesta permitió la especificación de guiones para razonar acerca de los campos del formulario utilizando una base de conocimiento rebatible, expresada en términos de un Programa Lógico Rebatible.

En síntesis, como se enunciara en los objetivos propuestos en la introducción de esta Disertación, puede concluirse que se ha abordado con éxito la caracterización del razonamiento con ontologías incompletas y potencialmente inconsistentes expresadas en Lógicas para la Descripción en términos de un sistema argumentativo como el proporcionado por la Programación en Lógica Rebatible en el marco de la iniciativa de la Web Semántica. Los principales resultados obtenidos pueden sintetizarse como sigue:

1. Se propuso un método para lidiar con definiciones de ontologías inconsistentes en la Web Semántica. Nuestra propuesta involucró corresponder ontologías expresadas en Lógicas para la Descripción con Programas Lógicos Rebatibles. Así, dada una consulta acerca de la pertenencia de una instancia a a un cierto concepto C expresada con respecto a una ontología con posibles inconsistencias, se realiza un análisis dialéctico para determinar todas las razones a favor y en contra de la plausibilidad de la afirmación $C(a)$.

2. La integración de datos es el problema de combinar datos residiendo en diferentes fuentes y el de proveer al usuario con una vista unificada de dichos datos. Como cuando se tienen varias ontologías, los conceptos allí definidos pueden tener nombres distintos para un mismo significado o aún nombres iguales para significados diferentes, para relacionar los conceptos entre dos ontologías diferentes se utilizaron reglas puente o como reglas de articulación. De esta manera, un concepto corresponde a una vista sobre otros conceptos de otra ontología. Se mostró también bajo qué condiciones la propuesta del razonamiento con δ-ontologías puede ser adaptado a los dos tipos de integración de ontologías *global-as-view* y *local-as-view* considerados en la literatura especializada.

3. Se analizaron las propiedades formales que se desprenden de este acercamiento novedoso al tratamiento de ontologías inconsistentes en la Web Semántica. Los principales resultados obtenidos fueron que, como la interpretación de δ-ontologías como Programas Lógicos Rebatibles es realizada a través de una función de transformación que preserva la semántica de las ontologías involucradas, los resultados obtenidos al consultar las operaciones sobre Aboxes son sensatas, consistentes y significativas.

4. Presentamos una propuesta para extender los formularios web para incorporar atributos rebatibles como parte del conocimiento que puede ser codificado por el diseñador del formulario. Vimos cómo dicho conocimiento puede ser especificado mediante un programa DeLP, y posteriormente, como una ontología expresada en Lógi-

cas para la Descripción. Así, la extensión propuesta permitió la especificación de guiones para razonar acerca de los campos del formulario utilizando una base de conocimiento rebatible, expresada en términos de un Programa Lógico Rebatible.

7.2. Trabajo Futuro

A partir del trabajo de investigación realizado durante el desarrollo de esta Tesis, se han obtenido distintos resultados que abren nuevas líneas de investigación. En particular, el estudio de la relación de la argumentación rebatible con las tecnologías de la Web Semántica abre un amplio espectro de aplicaciones incluyendo aplicaciones de comercio electrónico, implementación de sistemas multiagentes y redes sociales (Brickley and Miller, 2007).

Dentro de las limitaciones de nuestro acercamiento, como ya vimos en el Capítulo 4 podemos decir que el mismo no es capaz de lidiar con axiomas que requieran la construcción de reglas lógicas con disyunciones en la cabeza de las reglas, las cuales no son actualmente soportadas por la Programación en Lógica Rebatible. Una posible extensión a este trabajo consiste en mejorar a DeLP con la posibilidad de manejar disyunciones en la cabeza de las reglas. Los aportes del área de la Programación en Lógica Disyuntiva (Minker and Seipel, 2002) ciertamente podrían ser de ayuda en el ataque a estos problemas.

Otra arista asociada al acercamiento que necesita ser resuelta está dada por la correspondencia de axiomas de equivalencia de la Lógica para la Descripción en reglas de la Programación en Lógica Rebatible. Actualmente, un axioma de la forma "$C \equiv D$" genera dos reglas de la forma "$C(X) \prec D(X)$" and "$D(X) \prec C(X)$". A pesar de que las ontologías consideradas en esta Disertación modelan una gran parte de las ontologías, que no poseen este tipo de axiomas, esta situación claramente puede producir bucles durante la construcción de argumentos.

Actualmente, el estándar de la capa de Reglas de la Torta de la W3C se halla en discusión. Una posible extensión al trabajo presentado en esta Tesis puede consistir del estudio del problema sobre cómo el lenguaje X-DeLP podría ser adaptado al estándar RIF (por *Rule Interchange Format*), o, por el contrario, cómo extender la propuesta de RIF para representar rebatibilidad de reglas.

Bibliografía

Al-Asady, R. (1993), Reasoning with Exceptions: An Inheritance Based Approach, PhD thesis, University of Exeter.
Se introduce una técnica basada en estructuras de herencia múltiple que presenta razonamiento con herencia con excepciones sin la cual las redes de herencia producen excepciones. Se presentan además un número de propiedades que subsumen a formalismos como la Lógica Default, un algorimto y demostración de cómo la técnica presentada puede ser usada para especificar e implementar varios problemas de herencia no-monótona.

Alasoud, A., Haarslev, V. and Shiri, N. (2005), A Hybrid Approach for Ontology Integration, *in* 'VLDB Workshop on Ontologies-based techniques for DataBases and Information Systems (ODBIS)', Trondheim, Norway.
Proponen un marco para integración de ontologías el cual es un híbrido de vistas materializadas (data warehouses) y virtuales. Han desarrollado un prototipo del marco propuesto. Los autores afirman que, mientras tienen mucho trabajo por delante, sus experimentos indican que las ideas es dicho trabajo son promisorias y pueden resultar en contribuciones significativas tanto teóricas como prácticas.

Alchourron, C., Gärdenfors, P. and Makinson, D. (1985), 'On the logic of theory change: Partial meet functions for contraction and revision', *Journal of Symbolic Logic* (50), 510–530.

Ambler, S. W. (2006), 'UML 2 Activity Diagramming Guidelines'.
Presenta los elementos de los diagramas de actividad del lenguaje universal de modelado UML 2.0.

Antoniou, G. and Bikakis, A. (2007), 'DR-Prolog: A System for Defeasible Reasoning with Rules and Ontologies on the Semantic Web', *IEEE Transactions on Knowledge and Data Engineering* **19**(2), 233–245.
Proponen un sistema de razonamiento rebatible para la Web Semántica basado en su propio formalismo de razonamiento rebatible, llamado *Lógica Rebatible*. Notan que en la integración de ontologías, las inconsistencias surgen naturalmente, y en dicho marco, la lógica rebatible es muy útil para resolverlas. Su sistema es sintácticamente compatible con RuleML (RuleML,

2005), tiene reglas rebatibles y estrictas, está basado en una traducción a la programación en lógica con una sintaxis declarativa. Esta situación se puede caracterizar como *conflictos entre reglas*, los cuales se dan en los niveles de la capa de ontologías y en la capa de lógica y razonamiento. A nivel de la capa de ontologías, tienen: herencia *default* entre ontologías, mezcla de ontologías; y a nivel de la capa de lógica y razonamiento, tienen: reglas con excepciones como una manera natural de representar reglas de negocios, y razonamiento con información incompleta.

Antoniou, G., Bikakis, A. and Wagner, G. (2004*a*), A Defeasible Logic Programming System for the Web, *in* 'Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)', pp. 756–757.
Reportan la construcción de un sistema de razonamiento rebatible que puede llegar a ser utilizado como la capa de razonamiento de la Web Semántica. Reportan también que parte su trabajo ha sido la creación de un DTD para traducir teorías rebatibles en archivos XML. De hecho, su DTD es una extensión del DTD de la iniciativa *RuleML* (RuleML, 2005), que cubre reglas estrictas y rebatibles así como la relación de superioridad para reglas definida en su sistema de razonamiento rebatible.

Antoniou, G., Bikakis, A. and Wagner, G. (2004*b*), 'A System for Nonmonotonic Rules on the Web', *LNCS 3323 (Proc. of RuleML2004)* pp. 23–26.
Describe escenarios para el razonamiento con información incompleta e inconsistente útil para modelar integración de ontologías, reglas de negocios con excepciones. Además presenta un sistema para razonar en la Web sintácticamente compatible con RuleML, con reglas estrictas y rebatibles, basado en programación en lógica con semántica declarativa.

Antoniou, G., Billington, D., Governatori, G. and Maher, M. (2001), 'Representation results for defeasible logic', *ACM Trans. on Comp. Logic* **2**(2), 255–287.
Este artículo investiga las transformaciones y formas normales de la programación en lógica en el contexto de la *Defeasible Logic*, un formalismo de razonamiento no-monotónico basado en reglas y prioridades. Estas transformaciones han sido usadas en la implementación eficiente de la *Defeasible Logic*.

Antoniou, G. and van Harmelen, F. (2003), Web Ontology Language: OWL, Technical report, Department of Computer Science, University of Greece.
Se describe la motivación para OWL en términos de sus requerimientos, y su relación no trivial con RDF-Schema. Se describen los varios elementos de OWL con cierto detalle.

Asleson, R. and Schutta, N. (2005), *Foundations of Ajax*, A press, USA.
Presenta las extensiones Ajax al lenguaje de programación de guiones *Javascript*.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. and Patel-Schneider, P., eds (2003), *The Description Logic Handbook – Theory, Implementation and Applications*, Cambrid-

ge University Press.
Presenta la teoría de las *Lógicas para la Descripción* junto con sus extensiones y aplicaciones.

Baader, F. and Hanschke, P. (1991), A schema for integrating concrete domains into concept languages, *in* 'Proceedings of the Twelfth Joint Conference in Artificial Intelligence (IJCAI' 91)', pp. 452–457.
Presenta extensiones a las Lógicas para Descripción básicas (Baader et al., 2003) para representar y razonar con dominios concretos.

Baader, F. and Sattler, U. (1998), Description Logics with Aggregates and Concrete Domains: Part ii (extended) - ltcs report 98-02, Technical report, RWTH Aachen, LuFg Theoretische Informatik.
Extienden las Description Logics con dominios concretos (como números enteros y reales) y con funciones de agregación (como min, max, count, sum) que se hallan habitualmente en las sistemas de bases de datos.

Baral, C. (2003), *Knowledge Representation, Reasoning and Declarative Problem Solving*, Cambridge University Press.
Presenta el formalismo de la programación en lógica bajo semántica de conjuntos de respuesta (*answer sets*).

Bassiliades, N., Antoniou, G. and Vlahavas, I. (2004), A defeasible logic reasoner for the semantic web, *in* 'Proc. of the Workshop on Rules and Rule Markup Languages for the Semantic Web', pp. 49–64.
Describe un sistema llamado *DR-DEVICE* capaz de razonar sobre metadatos RDF sobre múltiples fuentes Web usando reglas rebatibles.

Bechhofer, S., Horrocks, I. and Patel-Schneider, P. F. (2003), 'Tutorial on OWL'.
URL: *http://www.cs.man.ac.uk/ horrocks/ISWC2003/Tutorial/*

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F. and Stein, L. A. (2004), 'OWL Web Ontology Language Reference'.
Presenta un detalle de todos los constructores del lenguaje de representación de la Web Semántica OWL.

Bench-Capon, T. J. M. and Dunne, P. E. (2007), 'Argumentation in artificial intelligence', *Artif. Intell.* **171**(10-15), 619–641.
Describen cómo un número de contribuciones fundacionales sentaron las bases para la formulación de los modelos de argumentación y su promoción en campos relacionados a la Inteligencia Artificial.

Berners-Lee, T. (2003), Foreword to "Spinning the Semantic Web", *in* 'Spinning the Semantic Web', The MIT Press, pp. i–xxiii.

Describe el paradigma de la web semántica. Para Berners-Lee, el uso de metadatos en en el contexto de la web semántica va a tener varios beneficios como: búsqueda más eficiente, validación en formularios web, los cachés de los proxies podrán determinar que los datos se distribuyen de acuerdo a los deseos de sus autores. El concepto más importante de la Web Semántica es que los documentos son entendibles por las máquinas y no solamente por los usuarios.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001), 'The Semantic Web', *Scientific American* **284**(5), 34–43.
Presenta una introducción al concepto de sistemas multiagentes con poder delegado por los usuarios para realizar citas con médicos utilizando como contexto la web semántica para obtener información sobre los mismos.

Boley, H., Dean, M., Grosof, B., Sintek, M., Spencer, B., Tabet, S. and Wagner, G. (2004), 'FOL RuleML: The First-Order Logic Web Language'.
Describe *FOL Rule ML* (*First-Order Logic RuleML*), el cual ha sido diseñado como el sublenguaje FOL (por Lógica de Primer Orden) de RuleML 0.9, el componente de reglas de SWRL FOL.

Boley, H., Hallmark, G., Kifer, M., Paschke, A., Polleres, A. and Reynolds, D. (2008), 'Core From RIF', Rule Interchange Format Working Group, W3C, MIT, ERCIM, Keio.
Este documento especifica RIF-Core, un subconjunto común de RIF-BLD y RIF-PRD basado en RIF-DTB 1.0.

Bondarenko, A., Dung, P., Kowalski, R. and Toni, F. (1997), 'An abstract argumentation-theoretic approach to default reasoning', *Artificial Intelligence 93:* pp. 63–101.

Borgida, A. (1996), 'On the Relative Expressiveness of Description Logics and Predicate Logics', *Artificial Intelligence* **82**(1–2), 353–367.
Muestran cómo los constructores usados en la literatura de las Lógicas para la Descripción pueden ser caracterizados exactamente como los predicados definibles por $\mathcal{L}_3$, el subconjunto de la Lógica de Primer Orden restringida a predicados monádicos y diádicos con tres símbolos de variable. Para representar restricciones numéricas usan *cuantificadores numéricos* y para representar roles que respetan la clausura transitiva utilizan disyunciones infinitas.

Brachman, R. (1979), 'On the epistemological status of semantic networks', *Associative Networks: Representation and Use of Knowledge by Computers* pp. 3–50.
Examina en detalle la historia de un conjunto de formalismos estructuras en redes para la representación de conocimiento, las llamadas *redes semánticas*.

Brena, R., Chesñevar, C. and Aguirre., J. (2006), Argumentation-supported information distribution in a multiagent system for knowledge management, *in* 'Proc. of ArgMAS

2005 (Utrecht, Netherlands, July 2005). In LNCS 4049, Springer Verlag', Springer Verlag, pp. 279–296.
Se explica cómo adaptar el marco multiagente JITIK para resolver el problema de distribuir información en grandes organizaciones. En el acercamiento basado en la Programación en Lógica Rebatible, los conflictos entre políticas de distribución entre usuarios son resueltos en base a un análisis dialéctico que determina qué usuarios recibirán qué pieza de información.

Brewka, G., Dix, J. and Konolige, K. (1997), *Non monotonic reasoning. An overview*, CSLI Publications, Stanford, USA.
Presenta un compendio de lógicas para razonar en forma no monótona.

Brickley, D. and Miller, L. (2007), 'FOAF Vocabulary Specification 0.91. Namespace Document 2 November 2007 - OpenID Edition'.
Esta especificación describe el lenguaje FOAF (por *Friend of A Friend*, o *Amigo de un Amigo*), definida como un diccionario de propiedades y clases nombradas usando la tecnología RDF de la W3C.
URL: *http://xmlns.com/foaf/spec/*

Calvanese, D., Giacomo, G. D. and Lenzerini, M. (2001), A Framework for Ontology Integration, *in* 'Proceedings of the 1st Semantic Web Working Symposium (SWWS 2001)'.
En el contexto de la integración de ontologías, donde una variedad de fuentes de información deben ser reconciliadas en términos de una ontología global, presentan el problema de la especificación de la correspondencia entre la ontología global y las ontologías locales.

Calvanese, D., Lenzerini, M. and Nardi, D. (1998), 'Description Logics for Conceptual Data Modeling', *Logics for Databases and Information Systems* .
Muestra cómo se pueden corresponder los modelos de datos entidad-relación y el orientado a objetos con las Lógicas para la Descripción. Se muestra cómo realizar los mapeos correspondientes y cómo tales mapeos preservan la semántica de los modelos originales.

Caminada, M. (2008), 'On the Issue of Contraposition of Defeasible Rules', *COMMA 2008* pp. 109–115.
Discute las condiciones bajo las que sería apropiado el uso de la contraposición (o *modus tollens*) de las reglas rebatibles. Postula que su uso depende del tipo de razonamiento considerado: *epistemológico* o *constitutivo*.

Carbogim, D., Robertson, D. and Lee, J. (2000), 'Argument-based applications to knowledge engineering', *The Knowledge Engineering Review* .
Presenta una descripción genérica de los sistemas argumentativos describiendo conceptualmente cada uno de sus elementos. Seguidamente, hace una recopilación de cómo se ha

aplicado la argumentación a los campos de toma de decisiones bajo incertidumbre, protocolos de negociación en sistemas multiagente y diseño de software.

Cecchi, L. A., Fillottrani, P. R. and Simari, G. R. (2006), On Complexity of DeLP through Game Semantics, *in* J. Dix and A. Hunter, eds, '11th. Intl. Workshop on Nonmonotonic Reasoning', pp. 386–394.
Realiza un análisis de la complejidad temporal de algunas de la tareas de razonamiento de DeLP. Los resultados principales son que el problema de determinar si un conjunto de reglas constituyen un argumento para un literal dado es P-completo, y que determinar si existe un argumento para un literal dado es NP.

Champin, P.-A. (2001), RDF Tutorial, Technical report, University of Lyon, France.
Presenta el lenguaje *Resource Description Framework*, recomendado por el *World Wide Web Consortium* para modelar los metadatos de los recursos de la Web.

Cheong, F. C. (1995), *Internet Agents: Spiders, Wanderers, Brokers, and Bots*, New Riders.
Presenta un breve resumen sobre el origen de la Web y discute los principales tipos de agentes para la Web.

Chesñevar, C., Brena, R. and Aguirre, J. (2005), Modelling power and trust for knowledge distribution: an argumentative approach, *in* 'LNAI Springer Series (Proc. of the 3rd Mexican International Conference on Artificial Intelligence – MICAI 2005)', Vol. 3789, pp. 98–108.
La confianza entre usuarios en el problema de la distribución de información se modela en un formalismo argumentativo, de tal manera que un proceso dialéctico se usa para decidir en presencia de políticas potencialmente conflictivas que modelan relaciones de poder y confianza.

Chesñevar, C. I., Maguitman, A. and Loui, R. (2000), 'Logical Models of Argument', *ACM Computing Surveys* **32**(4), 337–383.
Este survey presenta diferentes acercamientos relevantes para modelar la argumentación rebatible, así como la interrelación existente entre ellos.

Chesñevar, C. I., Simari, G. R. and García, A. (2000), Pruning Search Space in Defeasible Argumentation, *in* 'Proc. of the Workshop on Advances and Trends in Artificial Intelligence', XX International Conference of the SCCC – Santiago, Chile.
Se presenta un acercamiento basado en consistencia que permite podar el espacio de búsqueda en programación lógica rebatible.

Chesñevar, C., Simari, G. and Godo, L. (2005), 'Computing Dialectical Trees Efficiently in Possibilistic Defeasible Logic Programming', *LNAI Springer Series Vol. 3662 (Proc. of*

the 8th Intl. Conference on Logic Programming and Nonmonotonic Reasoning LPNMR 2005) pp. 158–171.
Adapta el algoritmo de poda de árboles dialécticos dela DeLP a P-DeLP, que es un formalismo que combina DeLP con un acercamiento cuantitativo para comparar argumentos en función de grados de certeza.

Clark, J. (1999), 'XSL Transformations (XSLT) Version 1.0. W3C Recommendation 16 Nov. 1999'.
Esta especificación define la sintaxis y semántica de XSLT, el cual es un lenguaje para transformar documentos XML en otros documentos XML.

Clark, P. (1990), Nonmonotonic Reasoning, Argumentation and Machine Learning, Technical report, TIMLG-38, Turing Institute, Glasgow, UK.
Este trabajo es uno de los precursores de la combinación de aprendizaje automatizado y argumentación. Presenta una discusión sobre las relaciones existentes entre el razonamiento no monótono, la argumentación y el aprendizaje automatizado.

Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D. L. and Stein, L. A. (2001), 'DAML+OIL (March 2001) Reference Description'.
DAML+OIL es un lenguaje de marcado semántico para recursos web. Se basa en RDF y RDF-Schema, y extiende estos lenguajes con primitivas más ricas.

Corcho, O., Gomez-perez, A., Leger, A., Rey, C. and Toumani, F. (2003), An ontology-based mediation architecture for e-commerce applications, *in* 'Proceedings of Intelligent Information Systems', Springer, pp. 477–486.
Presentan un marco de integración de mediación basado en ontologías para aplicaciones de comercio electrónico. El marco está basado en un acercamiento mediador/*wrapper* que soporta una vista integrada sobre múltiples fuentes de datos heterogéneas llenando el hueco entre las consultas de usuario y las ofertas específicas de los proveedores.

Cuenca Grau, B., Horrocks, I., Kazakov, Y. and Sattler, U. (2007), Just the right amount: extracting modules from ontologies, *in* 'WWW '07: Proceedings of the 16th international conference on World Wide Web', ACM, New York, NY, USA, pp. 717–726.
Para facilitar el reuso de ontologías, proponen un método para extraer una parte de una ontología de modo de garantizar la definición completa de un término.

Davis, R., Shrobe, H. and Szolovits, P. (1993), 'What is a Knowledge Representation', *AI Magazine* **14**(1), 17–33.
Definen la noción de *representación de conocimiento* (KR) y estudian los cinco roles de la misma: (i) la KR es un sustituto del objeto representado; (ii) la KR es un conjunto de compromisos ontológicos; (iii) la KR es una teoría fragmentaria del razonamiento inteligente;

(iv) la KR es un medio para la computación eficiente, y, (v) la KR es un medio para la expresión humana.

de Bruijn, J. (2005), 'Semantic Web: Lecture 7 - Description Logic Reasoning'.
Presenta una introducción a las Lógicas para la Descripción junto con ejemplos sobre el razonamiento con el método de tableaux.

Decker, S., Fensel, D., van Harmelen, F., Horrocks, I., Melnik, S., Klein, M. and Broekstra, J. (2000), 'Knowledge representation on the web', *Proc. of the 2000 Description Logic Workshop (DL 2000)* pp. 89–97.
Presenta el lenguaje de descripción de ontologías OIL cuya semántica está basada en lógicas para la descripción.

Dong, J., Lu, H., Lai, K. and Wang, S. (2004), 'XML-based decision support systems: Case study for portfolio selection', *International Journal of Information Technology & Decision Making* **3**(4), 651–662.
Enfatizan la adopción de la tecnología XML en el desarrollo de sistemas de soporte a la toma de decisiones basados en Web (en inglés, *web-enabled*) eficientes y flexibles. Basado en un caso de estudio para sistema de selección de carteras de inversión (en inglés, *portfolio*), exploran las aristas de diseño en la aplicación de XML para superar el problema de la heterogeneidad del intercambio de datos y el mantenimiento de varios modelos de optimización de portfolios.

Dou, D., McDermott, D. V. and Qi, P. (2005), 'Ontology translation on the semantic web.', *Journal of Data Semantics* **2**, 35–57.
Presentan un marco para mezcla de ontologías que utiliza *reglas puente*. El sistema utiliza un lenguaje de representación interno llamado *Web-PDDL* al que se traducen las ontologías expresadas en DAML+OIL, lo cual se implementa usando un motor de inferencia llamado *OntoEngine*.

Dubinko, M., Klotz, L., Merrick, R. and Raman, T. (2003), 'XForms 1.0 - W3C Recommendation. 14 Oct. 2003'.
XForms es una aplicación XML que representa la siguiente generación de formularios para la Web. Parte los tradicionales formularios XHTML en tres partes: el modelo XForm, los datos de instancia y la interfaz de usuario. De esta manera, se separa la presentación del contenido, pemitiendo reuso, tipado fuerte, independencia del dispositivo, necesidad de *scripting* reducida y pocos viajes al servidor.

Dung, P. M. (1993*a*), An argumentation semantics for logic programming with explicit negation, *in* 'Proceedings ICLP'93', MIT Press, pp. 616–630.

Dung, P. M. (1993*b*), On the aceptability of arguments and its fundamental role in non-monotonic reasoning and logic programming by n-persons games, *in* 'Proceedings of the 13th International Joint Conference in Artificial Intelligence (IJCAI)', pp. 852–857.
Estudia el proceso de argumentación y la plausibilidad de su implementación en computadoras. Se presenta la relación de un marco de argumentación con otros formalismos de razonamiento no-monótono.

Dwight, J., Erwin, M. and Niles, R. (1997), *Using CGI. Second Edition*, QUE Corp.
Presenta los fundamentos de la programación de aplicaciones web con procesamiento de datos de formularios web usando CGI (*Common Gateway Interface*).

Eiter, T., Lukasiewicz, T., Schindlauer, R. and Tompits, H. (2004), 'Combining Answer Set Programming with Description Logics for the Semantic Web', *KR 2004* pp. 141–151.
Para integrar reglas y ontologías en la Web Semántica, proponen una combinación de programación en lógica bajo semántica de conjuntos de respuesta con las Lógicas para la Descripción $\mathcal{SHIF}(D)$ y $\mathcal{SHOIN}(D)$, las cuales subyacen a OWL Lite y OWL DL, resp. Esta combinación permite construir reglas sobre ontologías.

Falappa, M. A., Kern-Isberner, G. and Simari, G. R. (2002), 'Explanations, Belief Revision and Defeasible Reasoning', *Artificial Intelligence* **141**, 1–28.
Presentan diferentes construcciones para revisión de creencias no-priorizada y relacionan los operadores de revisión formulados con la Programación en Lógica Rebatible.

Fensel, D., Bussler, C., Ding, Y., Kartseva, V., Klein, M., Korotkiy, M., Omelayenko, B. and Siebes, R. (2002), 'Semantic Web Application Areas'.
Se presentan las áreas de aplicación de la Web Semántica, con énfasis en los campos de gestión de conocimiento, comercio electrónico y servicios web.

Fermé, E. (2005), 'Revisión de Creencias'.
Presenta una introducción al tema Revisión de Creencias, en particular al modelo AGM. También presenta consideraciones sobre la implementación de las operaciones de cambio.

Flouris, G. (2006), On Belief Change and Ontology Evolution, PhD thesis, Department of Computer Science, University of Crete.
Estudia los escenarios para la evolución de ontologías (es decir, hacer cambios a una ontología en función de cambios en el dominio de aplicación) expresadas en Lógicas para la Descripción utilizando el modelo AGM.

Frakes, W.; Baeza-Yates, R. (1992), *Information Retrieval. Data Structures & Algorithms*, Prentice Hall.

Presenta una descripción de todos los conceptos que comprenden la recuperación de información, desde estructuras de datos, técnicas de indexación, algoritmos y métodos para medir la performance de un sistema de recuperación de información.

García, A. J. (1997), Programación en lógica rebatible: su definición, Master's thesis, Departamento de Cs. de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina.
Presenta la Programación en Lógica Rebatible, un formalismo de razonamiento no-monótono basado en la argumentación rebatible. Presenta además la implementación de la misma usando la JAM (*Justication Abstract Machine), que es una extensión de la WAM (Warren Abstract Machine), máquina subyacente de los intérpretes del lenguaje Prolog.*

García, A. J. and Simari, G. R. (2004), 'Defeasible Logic Programming: An Argumentative Approach', *Theory and Practice of Logic Programming* **4**(1), 95–138.
Presenta la Programación en Lógica Rebatible junto con la teoría de argumentación rebatible que la sustenta.

Gómez, S. A. (2003), Integración de Técnicas de Aprendizaje Automatizado con Sistemas Argumentativos, Master's thesis, Universidad Nacional del Sur.
Presenta un acercamiento híbrido en que un programa lógico rebatible se utiliza para revisar la salida de una red neuronal del modelo de la Teoría de la Resonancia Adaptativa Difusa utilizada para realizar agrupación de patrones. Este acercamiento permite resolver los casos donde la salida de la red neuronal es ambigua.

Gómez, S. A. and Chesñevar, C. I. (2004), A Hybrid Approach to Pattern Classification Using Neural Networks and Defeasible Argumentation, *in* 'Proc. of 17th Intl. FLAIRS Conference. Miami, Florida, USA', American Assoc. for Art. Intel., pp. 393–398.

Gómez, S. A., Chesñevar, C. I. and Simari, G. R. (2005*a*), A First Approach to Combining Ontologies and Defeasible Argumentation for the Semantic Web, *in* 'XI Congreso Argentino en Ciencias de la Computación (CACIC 2005). Concordia, Argentina', pp. 1–12.

Gómez, S. A., Chesñevar, C. I. and Simari, G. R. (2005*b*), 'Embedding defeasible argumentation in the Semantic Web: an argument-based approach', *VII Workshop de Investigadores en Ciencias de la Computación* pp. 153–157.

Gómez, S. A., Chesñevar, C. I. and Simari, G. R. (2005*c*), 'Incorporating Defeasible Knowledge and Argumentative Reasoning in Web-based Forms', *Third Workshop of Intelligent Techniques for Web Personalization (ITWP'05), International Joint Conference in Artificial Intelligence (IJCAI'05)* pp. 9–16.

Gómez, S. A., Chesñevar, C. I. and Simari, G. R. (2006*a*), An Approach to Handling Inconsistent Ontology Definitions based on the Translation of Description Logics into Defeasible Logic Programming, *in* 'Proc. of the XII Congreso Argentino de Ciencias de la Computación (CACIC'06)', pp. 1185–1196.

Gómez, S. A., Chesñevar, C. I. and Simari, G. R. (2006*b*), Problems and Challenges for Ontology Integration in the Semantic Web, *in* J. S. Ierache, ed., 'Anales del Octavo Workshop de Investigadores en Ciencias de la Computación (WICC 2006)', pp. 69–73.

Gómez, S. A., Chesñevar, C. I. and Simari, G. R. (2007*a*), Hacia una Integración de Argumentación Rebatible y Ontologías en la Web Semántica, *in* Z. R. et al., ed., 'Anales del IX Workshop de Investigadores en Ciencias de la Computación (WICC 2007) - 1a ed.', pp. 91–95.

Gómez, S. A., Chesñevar, C. I. and Simari, G. R. (2007*b*), 'Inconsistent Ontology Handling by Translating Description Logics into Defeasible Logic Programming', *Revista Iberoamericana de Inteligencia Artificial* **11**(35), 11–22.

Gómez, S. A., Chesñevar, C. I. and Simari, G. R. (2008*a*), An Argumentative Approach to Reasoning with Inconsistent Ontologies, *in* T. Meyer and M. A. Orgun, eds, 'Proc. of the Knowledge Representation in Ontologies Workshop (KROW 2008)', Vol. CPRIT 90, Sydney, Australia, pp. 11–20.

Gómez, S. A., Chesñevar, C. I. and Simari, G. R. (2008*b*), 'Defeasible Reasoning in Web Forms Through Argumentation', *International Journal of Information Technology & Decision Making* **7**, 71–101.

Gómez, S. A., Chesñevar, C. I. and Simari, G. R. (2008*c*), Hacia la Solución del Problema de Chequeo de Instancia en Ontologías Inconsistentes Usando Argumentación Rebatible, *in* 'X Workshop de Investigadores en Ciencias de la Computación 2008 (WICC 2008)', pp. 66–70.

Gómez, S. A., Chesñevar, C. I. and Simari, G. R. (2008*d*), Integration of Web-based Forms with Ontologies in the Semantic Web, *in* A. C. L. Norberto Caminoa, Fernanda Carmona, ed., 'Actas del XIV Congreso Argentino de Ciencias de la Computación (CACIC 2008)', Chilecito, Argentina, 6-10 octubre 2008.

Grosof, B., Horrocks, I., Volz, R. and Decker, S. (2003), 'Description Logic Programs: Combining Logic Programs with Description Logics', *WWW2003, Budapest, Hungary, ACM 1-58113-680-3/03/0005* .
Presentan una función de traducción para traducir una parte de las Lógicas para la Descripción en Programas Lógicos.
URL: *http://www.www2003.org/cdrom/papers/refereed/p117/p117-grosof.html*

Gruber, T. R. (1993), 'A translation approach to portable ontologies', *Knowledge Acquisition* **5**(2), 199–220.
Describe un mecanismo para definir ontologías que son portables entre diferentes sistemas de representación. Las definiciones escritas en cálculo de predicados son traducidas a un sistema llamado *Ontolingua*, que utiliza marcos (*frames*).

Guan, S. and Mcmullen, P. (2005), 'Organizing information on the next generation web – design and implementation of a new bookmark structure', *International Journal of Information Technology and Decision Making* **4**(5), 97–116.
Explora el problema de crear índices para objetos web, muestra un prototipo para compartir *bookmarks* entre usuarios.

Gulbransen, D. and Rawlings, K. (1998), *HTML Dinámico. Edición Especial*, QUE, Prentice Hall.
Presenta un conjunto de técnicas de programación para combinar HTML, JavaScript, Hojas de Estilo en Cascada para la programación de aplicaciones web.

Haarslev, V. and Möller, R. (2001), RACER System Description, Technical report, University of Hamburg, Computer Science Department.
Presenta una breve introducción al motor de inferencias para Lógicas de la Descripción llamado Racer (véase la entrada (Haarslev and Möller, 2004)).

Haarslev, V. and Möller, R. (2004), RACER User's Guide and Reference Manual Version 1.7.19, Technical report, University of Hamburg, Computer Science Department.
Manual de uso de la herramienta de razonamiento con Lógicas para la Descripción RACER. Las bases de conocimiento u ontologías se describen mediante una sintaxis *LISP-like*. Se explica cómo definir ontologías, realizar consultas e interpretar sus resultados ya sea por medio de guiones invocando a la herramienta en la línea de comandos o como servidor a través de la interfaz para el lenguaje Java *TCP Socket* llamada *JRacer*. Probamos la herramienta para la plataforma *Windows XP*.

Haase, P. and Motik, B. (2005), A mapping system for the integration of owl-dl ontologies, *in* A. Hahn, S. Abels and L. Haak, eds, 'IHIS 05: Proceedings of the first international workshop on Interoperability of heterogeneous information systems', ACM Press, pp. 9–16.
Para permitir la interoperabilidad entre aplicaciones en sistemas distribuidos de información basados en ontologías heterogéneas, es necesario definir formalmente la noción de un mapeo entre ontologías. En (Haase and Motik, 2005), Haase y Motik definen sistema de mapeos para ontologías OWL-DL, donde los mapeos son expresadas como correspondencias entre consultas conjuntivas sobre ontologías. Como contestar consultas sobre tal sistema de mapeo general es no decidible, ellos identifican un fragmento decidible del sistema de mapeo, que

corresponde a OWL-DL extendido con *reglas seguras respecto de DL*. También muestran cómo el sistema de mapeo puede ser aplicado a la tarea de la integración de ontologías y presentan un algoritmo para resolver consultas. .
URL: *http://www.aifb.uni-karlsruhe.de/WBS/pha/publications/owlmapping05ihis.pdf*

Heflin, J., Hendler, J. and Luke, S. (2003), SHOE: A Blueprint for the Semantic Web, *in* D. Fensel, J. Hendler, H. Lieberman and W. Walhster, eds, 'Spinning the Semantic Web', The MIT Press, pp. 29–63.
Presentan un lenguaje para extender HTML con definiciones de ontologías. Presentan la sintaxis y semántica.

Heymans, S. and Vermeir, D. (2002), A Defeasible Ontology Language, *in* 'CoopIS/DOA/ODBASE', pp. 1033–1046.
Presentan un enfoque argumentativo a las Lógicas para la Descripción.

Horrocks, I. (1998), The FaCT System, *in* 'TABLEAUX '98: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods', Springer-Verlag, London, UK, pp. 307–312.
FaCt es un clasificador de Lógicas para la Descripción que ha sido implementado como un *test-bed* para un sistema de satisfacibilidad basado en el método de *tableaux* para calcular subsunción de clases.

Horrocks, I. and Patel-Schneider, P. F. (2003), 'A Proposal for an OWL Rules Language. Semantics and Abstract Syntax'.
Es una descripción de una extensión basada en reglas al lenguaje de ontologías web OWL. Incluye una sintaxis de alto nivel para cláusulas de Horn en los sublenguajes OWL DL y OWL Lite. La semántica de este lenguaje es model-theoretic, la cual se usa para dar significado a ontologías OWL incluyendo las reglas en una sintaxis abstracta. Se presentan una sintaxis basada en OWL XML junto con un mapeo a grafos RDF basada en la sintaxis de intercambio OWL RDF/XML.

Horrocks, I. and Patel-Schneider, P. F. (2004), 'Reducing OWL Entailment to Description Logic Satisfiability', *Journal of Web Semantics* **1**(4), 345–357.
Muestran la reducción del problema de la inferencia en los lenguajes OWL-DL y OWL-Lite al problema de la satisfacibilidad de bases de conocimiento en las Lógicas para la Descripción $\mathcal{SHOIN}(D)$ y $\mathcal{SHIF}(D)$, respectivamente.

Horrocks, I. and Sattler, U. (1999), 'A Description Logic with Transitive and Inverse Roles and Role Hierarchies', *Journal of Logic and Computation* **9**(3), 385–410.
Se presentan algoritmos de tablaux para decir la satisfacibilidad de conceptos y subsunción en Lógicas para la Descripción que extienden $\mathcal{ALC}$ con roles transitivos e inversos. Los algoritmos presentados en este trabajo son adecuados para propósitos de implementación.

Horrocks, I. and Tessaris, S. (2000), A Conjunctive Query Language for Description Logic Aboxes, *in* 'AAAI-00 Proceedings'.
Presentan una técnica novedosa que puede ser utilizada para proveer un lenguaje de consultas expresivo para sistemas de representación de conocimiento basados en Lógicas para la Descripción. Esta técnica está baada en la reducción a la satisfacibilidad de la base de conocimiento, por lo cual puede ser adaptada a las implementaciones de razonadores actuales y futuros.

Horty, J. F., Thomasson, R. H. and Touretzky, D. S. (1990), 'A skeptical theory of inheritance in nonmonotonic semantic networks', *Artificial Intelligence* (42), 311–348.
Describe un acercamiento para el razonamiento con herencia en redes semánticas con herencia múltiple con excepciones.

Huang, Z., van Harmelen, F. and ten Teije, A. (2005), Reasoning with inconsistent ontologies, *in* L. P. Kaelbling and A. Saffiotti, eds, 'Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)', Edinburgh, Scotland, pp. 454–459.
Presentan un marco para razonar con ontologías inconsistentes. Este trabajo tiene un enfoque basado en razonamiento no monótono; en particular, se basa en lógica para-consistente y revisión de creencias. Sin embargo, funciones de selección predefinidas son utilizadas para lidiar con relevancia de conceptos. De esta manera, crean órdenes específicos (dependientes de las consultas particulares), enfoque que acelera la resolución de consultas. Introducen también las definiciones formales de las funciones de selección e investigan las estrategias de procesamiento del razonamiento con inconsistencia basados en una estrategia lineal y proveen una implementación llamada *PION*.

Huang, Z., van Harmelen, F., ten Teije, A., Groot, P. and Visser, C. (2004), Reasoning with Inconsistent Ontologies: a general framework, Technical report, Department of Artificial Intelligence. Vrije Universiteit Amsterdam. Document ID: SEKT/2004/D3.4.1.1/v1.0
Este reporte técnico es una versión previa de (Huang et al., 2005).

Hunhs, M. and Singh, M. P. (1998), *Readings in agents*, Morgan Kaufmann.
Presenta una recopilación de artículos sobre el campo de los agentes de software.

Kakas, A. and Toni, F. (1999), 'Computing argumentation in logic programming', *Journal of Logic and Computation* **9**(4), 515–562.
Desarrollan un marco computacional abstracto en el cual varias semánticas de argumentación pueden ser computadas variando parámetros de una teoría de prueba básica.

Klein, M. (2001), Combining and relating ontologies: an analysis of problems and solutions, *in* A. Gomez-Perez, M. Gruninger, H. Stuckenschmidt and M. Uschold, eds, 'Workshop on Ontologies and Information Sharing, IJCAI'01', Seattle, USA.

Presenta un resumen de los acercamientos para la integración de ontologías y da un marco en el cual se unifican las convenciones encontradas en la literatura.
URL: *citeseer.ist.psu.edu/klein01combining.html*

Koivunen, M.-R. and Miller, E. (2001), 'W3C Semantic Web Activity', Proceedings of the Semantic Web Kick-off Seminar in Finland Nov 2.
Presenta una descripción de los objetivos de estandarización de la organización W3C con respecto a la problemática de la Web Semántica.

Krause, P., Ambler, S., Elvang-Goransson, M. and Fox, J. (1995), 'A logic of argumentation for reasoning under uncertainty'.
Presentan una teoría de prueba sintáctica y semántica para argumentación. En esta lógica llamada LA, las proposiciones son etiquetadas con los argumentos que soportan su validez. Los argumentos se pueden agregar para coleccionar más información acerca de la validez de las proposiciones de interés; de esta manera se pueden incorporar técnicas para asignar grados de confianza símbolicos y numéricos a una afirmación.
URL: *citeseer.nj.nec.com/krause95logic.html*

Krötzch, M., Rudolph, S. and Hitzler, P. (2006), Complexity of Horn Description Logics, *in* B. Cuenca Grau, P. Hitzler, C. Shankey and E. Wallace, eds, 'Proceedings of the 2nd Workshop on OWL: Experiences and Directions (OWLED-06). CEUR Workshop Proceedings 2006'.
Horn-$\mathcal{SHIQ}$ ha sido identificada como un fragmento de la Lógica para la Descripción $\mathcal{SHIQ}$ para la cual la inferencia se halla en PTIME con respecto al tamaño de la Abox. En este trabajo, se muestra que Horn-$\mathcal{SHIQ}$ es EXPTIME-hard con respecto al tamaño de la base de conocimiento.

Laera, L., Blacoe, I., Tamma, V., Payne, T., Euzenat, J. and Bench-Capon, T. (2007), Argumentation over ontology correspondences in MAS, *in* 'AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems', ACM, New York, NY, USA, pp. 1–8.
Presentan un acercamiento para soportar la creación o el rechazo de argumentos que soportan o rechazan correspondencias entre ontologías en el contexto de un sistema multiagente donde los agentes interactúan para llegar a un acuerdo sobre el significado de términos en las ontologías de cada uno.

Laera, L., Tamma, V., Euzenat, J., Bench-Capon, T. and Payne, T. (2006), Reaching agreement over ontology alignments, *in* 'Proceedings of the 5th International Semantic Web Conference (ISWC 2006), Athens, GA', Vol. 4273/2006, Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

Presentan un acercamiento para soportar la creación o el rechazo de argumentos que soportan o rechazan correspondencias entre ontologías.

Lam, S. C., Pan, J. Z., Sleeman, D. and Vasconcelos, W. (2006), 'A Fine-Grained Approach to Resolving Unsatisfiable Ontologies', *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence* pp. 428–434.
Presentan una técnica para hallar los axiomas que producen inconsistencias en un ontología. La técnica permite hallar cuáles son los conceptos de la ontología responsables de la inconsistencia.

Lenat, D. (1995), 'CYC: A Large-Scale Investment in Knowledge Infrastructure', *Communications of the ACM* **38**(11), 33–38.
Presenta el proyecto CYC, un esquema universal de 10^5 conceptos generales acerca de la realidad humana.

Lenzerini, M. (2002), 'Data integration: A theoretical perspective', *Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2002, Madison, Winsconsin, USA* .
La integración de datos es el problema de combinar datos provenientes de diferentes fuentes, proveyendo al usuario de una vista unificada de dichos datos. Este artículo pone énfasis en: modelar una aplicación de integración de datos, procesar consultas en una aplicación de integración de datos, lidiar con fuentes de datos inconsistentes, y razonar sobre consultas.

Li, L., Wu, B. and Yang, Y. (2005), Agent-based Ontology Integration for Ontology-based Applications, *in* 'Australasian Ontology Workshop (AOW 2005), Jointly held with the 18th Australian Joint Conference on Artificial Intelligence, Conference in Research and Practice in Information Technology (CRPIT) series', Vol. 58, Australian Computer Society, pp. 53–59.
Se desarrolla un marco de integración de ontologías basado en agentes que consumen ontologías en aplicaciones basadas en ontologías así como para agentes involucrados en tareas de integración de ontologías. Una cualidad de su método es que las ontologías derivadas pueden ser reusadas. También, presentan un prototipo construido utilizando la plataforma *JADE* para la evaluación del método propuesto.

Lie, H. W. and Bos, B. (1996), 'Cascading Style Sheets, level 1. W3C Recommendation 17 Dec 1996'.
Este documento especifica el mecanismo de nivel 1 de las Hojas de Estilo en Cascada (CSS). CSS1 es un mecanismo simple que permite a los autores y lectores adjuntar estilo (*e.g.*, fuentes, colores y espaciado) a documentos HTML. El lenguaje CSS1 es legible y producible por humanos y expresa el estilo en una terminología de publicación.

Lloyd, J. (1987), *Foundations of Logic Programming.*, Springer-Verlag.
Presenta el algoritmo de resolución formalmente junto con un desarrollo también formal de la fundamentación matemática de la programación en lógica.

Lloyd, W. and Topor, R. (1984), 'Making PROLOG More Expressive', *Journal of Logic Programming* **1**(3), 225–240.
Se presentan las transformaciones de Lloyd-Topor para programas Prolog.

Lutz, C. (2003), 'Description Logics with Concrete Domains—A Survey', *Advances in Modal Logic* **4**, 265–296.
Se da un resumen de las Lógicas para la Descripción con dominios concretos y se resumen resultados de decidibilidad y complejidad de la literatura.

Lutz, C., Areces, C., Horrocks, I. and Sattler, U. (2004), 'Keys, nominals, and concrete domains', *Journal of Artificial Intelligence Research* .
Presentan una extensión a las Lógicas para la Descripción con dominios concretos para representar lo que los autores llaman *restricciones de clave* (en inglés, *key constraints*), lo cual permite la expresión de sentencias tales como "los ciudadanos norteamericanos están unívocamente identificados por su número de seguro social". Basados en esta idea, introducen un número de Lógicas para la Descripción y realizan un análisis detallado de su decidibilidad y complejidad computacional.

Manola, F. and Miller, E. (2004), 'Rdf primer'.
Presenta los conceptos introductorios para comprender el Resource Description Framework (RDF). RDF permite definir relaciones entre objetos en la web semántica en la forma de sentencias con la forma sujeto-predicado-objeto. RDF es un lenguaje de representación de conocimiento orientado al intercambio de información entre aplicaciones más que a la presentación de información para humanos. La sintaxis de RDF está basada en XML y los objetos que referencia son denotados utilizando URIs. RDF utiliza un esquema particular llamado XML/RDF donde los argumentos de los marcadores se consideran en formato UNICODE, esto permite representar información en muchos lenguajes humanos. RDF puede verse como un digrafo donde por cada sentencia el sujeto y el objeto se representan como nodos y el predicado como un arco del sujeto al objeto. Los nodos pueden ser literales (cajas) u objetos (elipses). Los grafos se pueden representar también como tripletas de sujeto-predicado-objeto, donde cada tripleta corresponde a un arco, lo que hace secundario al gráfico de un grafo.
URL: *http://www.w3.org/TR/2004/REC-rdf-primer-20040210/*

McCarthy, J. (1990), 'Formalizing Common Sense: Papers by John McCarthy'.
Extendiéndose sobre un período de 30 años, esta es una colección de artículos escritos por John McCarthy sobre Inteligencia Artificial.

McCarthy, J. and Hayes, P. J. (1969), Some Philosophical Problems from the Standpoint of Artificial Intelligence, *in* B. Meltzer and D. Mitchie, eds, 'Machine Intelligence 4', Edinburgh University Press, pp. 463–502.
En este trabajo se formula por vez primera el problema de la cualificación como limitante en toda formalización de razonamiento de sentido común.

McGrath, S. (1998), *XML by example. Building e-commerce applications*, Prentice Hall.
Presenta los fundamentos del lenguaje de marcado XML.

McGuiness, D., Fikes, R., Stein, L. A. and Hendler, J. (2003), DAML-ONT: An Ontology Language for the Semantic Web, *in* D. Fensel, J. Hendler, H. Lieberman and W. Walhster, eds, 'Spinning the Semantic Web', The MIT Press, pp. 65–93.
El lenguaje de marcado para agentes de DARPA (DARPA Agent Markup Language, DAML) es un lenguaje de ontologías orientado a la definición de clases, subclases, sus propiedades, restricciones y descripciones individuales de objetos. El lenguaje posee otra porción, llamada *DAML-L*, orientada a la codificación de reglas de inferencia. La semántica de DAML se da en términos de KIF, cuya semántica está definida en términos de lógica de primer orden.

McGuiness, D. L. and van Harmelen, F. (2004), 'OWL Web Ontology Language Overview'.
Presenta un detalle de todos los constructores del lenguaje de representación de la Web Semántica OWL.
URL: *http://www.w3.org/TR/owl-features/*

Meyer, T., Lee, K. and Booth, R. (2005), 'Knowledge Integration for Description Logics'.
Modifican técnicas para la gestión de inconsistencia proposicional para manejar inconsistencia en el marco de las DLs. Muestran cómo la estructura adicional ofrecida por las DLs puede ser usada para refinar dichas técnicas. Su enfoque es en la semántica formal para dichas técnicas, también presentan procedimientos de decisión en alto nivel para las estrategias de integración discutidas.

Meyer, T., Lee, K., Booth, R. and Pan, J. Z. (2006), Finding maximally satisfiable terminologies for the Description Logic $\mathcal{ALC}$, *in* 'Proceedings of AAAI-06'.
Para ontologías representadas como Tboxes en Lógicas para la Descripción (DL), los razonadores DL optimizados son capaces de detectar errores lógicos, pero existe poco soporte para resolver tales problemas. Un posible remedio consiste en debilitar la información disponible de tal manera que los errores desaparezcan, pero limitando el proceso lo más posible. En este artículo, se propone un procedimiento basado en *tableau* para hallar las terminologías maximalmente satisfacibles.

Miller, R. (2006), 'Practical UML: A Hands-On Introduction for Developers'.

Presenta los elementos de los diagramas de actividad del lenguaje universal de modelado UML 2.0.

Minker, J. and Seipel, D. (2002), 'Disjunctive Logic Programming: A Survey and Assessment', *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski* pp. 472–511.
Describe los campos de la programación en lógica disyuntiva y las bases de datos disyuntivas.

Minsky, M. (1975), A framework for representing knowledge, *in* P. Winston, ed., 'The Psychology of Computer Vision', McGraw-Hill.
Describe los sistemas de marcos (*frames*) como un formalismo para la representación de conocimiento y luego se concentra en cuál debería ser el contenido del conocimiento dependiendo del dominio. Discute el enorme problema del volumen de conocimiento de sentido común necesario para entender textos simples expresados en lenguaje natural y sugiere que las redes de marcos son un acercamiento apropiado para tal problema. Argumenta que los acercamientos numéricos son inherentemente limitados. Evidentemente, este artículo fue publicado en pleno debate en el campo de la Inteligencia Artificial sobre la adecuación del modelo del razonamiento humano en función del razonamiento simbólico (marcos) versus el razonamiento numérico (redes neuronales incipientes como los perceptrones de McCulloch & Pitts), y queda muy claro cuál es la posición de Minsky. .

Mitra, P. (2004), An Algebraic Framework for the Interoperation of Ontologies, PhD thesis, Department of Electrical Engineering.
Mitra presenta un *álgebra de composición de ontologías* (*Ontology-Composition Algebra*) que consiste de un conjunto de operadores básicos que pueden ser usados para manipular ontologías. Mitra ha desacoplado la maquinaria algebráica que es usada para manipular ontologías de la componente que deriva la correspondencia semántica entre ontologías para crear dos componentes distinguidas: (1) las *funciones de generación de reglas de articulación* generan reglas de articulación entre pares de ontologías, y (2) los *operadores algebraicos* usan las reglas de articulación para componer las ontologías fuente.

Mitra, P. and Wiederhold, G. (2001), 'An Algebra for Semantic Interoperability of Information Sources', *Proceedings of the 2nd IEEE International Symposium on Bioinformatics and Bioengineering* pp. 174–183.
Presenta el sistema ONION que permite la interoperación semántica entre varias ontologías articulándolas. Una articulación se centra en la intersección semántica relevante entre dos ontologías.

Mitra, P. and Wiederhold, G. (2002), Resolving Terminological Heterogeneity in Ontologies, pp. 45–50.

Presenta un sistema que permite la interoperación entre fuentes de información usando ontologías. Se presentan dos métodos para aparear métodos usados en diferentes terminologías basados en similitudes lingüisticas entre sus términos. El primer método utiliza un diccionario como Wordnet y el segundo determina similitudes de palabras tomadas de cuerpo de documentos específicos del dominio.

Mitra, P., Wiederhold, G. and Kersten, M. (2000), 'A Graph-Oriented Model for Articulation of Ontology Interdependencies', *Lecture Notes in Computer Science* **1777**, 86+.
Presenta el sistema ONION para componer ontologías asistiendo a los usuarios a lograr este objetivo en forma semi-automatizada.
URL: *citeseer.ist.psu.edu/mitra00graphoriented.html*

Motik, B., Sattler, U. and Studer, R. (2004), Query answering for OWL-DL with rules, *in* 'Proceedings of the 3rd International Semantic Web Conference (ISWC 2004)'.
Presenta una combinación del lenguaje de ontologías OWL-DL con reglas. La combinación presentada es decidible a través de la restricción de las reglas consideradas a las *reglas seguras*, requiriendo que cada variable de una regla aparezca en el cuerpo de la misma.

Motik, B., Volz, R. and Bechhofer, S. (2003), 'DLP - Description Logic Programs'.
DLP es un sistema que permite convertir ontologías OWL-DL en programas lógicos disyuntivos.
URL: *http://kaon.semanticweb.org/alphaworld/dlp*

Motik, B., Vrandecić, D., Hitzler, P., Sure, Y. and Studer, R. (2005), dlpconvert: Converting owl dlp statements to logic programs, *in* 'European Semantic Web Conference 2005 (ESWC 2005) Demos and Posters'.
Presenta el sistema *dlpconvert* que permite obtener un programa Prolog a partir de una ontología expresada en OWL.

Nardi, D. and Brachman, R. J. (2003), An Introduction to Description Logics, *in* '(Baader et al., 2003)', chapter 1, pp. 5–44.
Presenta las Lógicas para la Descripción, un formalismo de representación de conocimiento basado en conceptos (predicados unarios) y relaciones entre conceptos (predicados binarios) que formaliza las redes de marcos (*frames*).

Noy, N. F. (2004), 'Semantic integration: a survey of ontology-based approaches', *SIGMOD Rec.* **33**(4), 65–70.
Este artículo brinda una recopilación de acercamientos a la integración semántica de ontologías. Se discuten las técnicas para hallar correspondencias entre ontologías, formas declarativas de representar tales correspondencias y el uso de tales correspondencias en tareas de integración semántica.

Nute, D. (1988), Defeasible Reasoning, *in* J. H. Fetzer, ed., 'Aspects of Artificial Intelligence', Kluwer Academic Publishers, Norwell, MA, pp. 251–288.

OMG (2007), 'OMG Unified Modeling Language (OMG UML) Infrastructure, V2.1.2'.
El *Lenguaje Unificado de Modelado* (*UML*, en inglés, *Unified Modeling Language*) es un lenguaje de modelado de sistemas de software gráfico para visualizar, especificar, construir y documentar un sistema.

Parsia, B. and Sirin, E. (2004), Pellet: An OWL DL Reasoner, *in* '3rd International Semantic Web Conference (ISWC2004)'.
Los autores afirman que los razonadores estándar para DL son insuficientes para OWL ya que no son capaces de manejar situaciones tales como razonamiento con individuos, eliminación de la restricción de nombres únicos, razonamiento con nominales. Con estos objetivos en vista, desarrollaron a *Pellet*, un razonador para OWL DL.

Parsons, S., Sierrra, C. and Jennings, N. (1998), 'Agents that Reason and Negotiate by Arguing', *Journal of Logic and Computation* **8**, 261–292.
La necesidad para la negociación en sistemas multiagentes se basa en la necesidad de los agentes de resolver problemas a partir de su interdependencia. La negociación brinda una solución a estos problemas al darle a los agentes los medios para resolver sus objetivos conflictivos y corregir inconsistencias en su conocimiento de la visión del mundo de otros agentes. Proponen un marco basado en argumentación que permite a los agentes negociar para establecer maneras apropiadas de resolver problemas.

Pemberton, S., Austin, D., Axelsson, J., Çelik, T., Dominiak, D., Elenbaas, H., Epperson, B., Ishikawa, M., Matsui, S., McCarron, S., Navarro, A., Peruvemba, S., Relyea, R., Schnitzenbaumer, S. and Stark, P. (2002), 'XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)'.
Esta especificación define XHTML 1.0 como una reformulación de HTML 4 como una aplicación XML 1.0.

Pinto, H. S., Gómez-Pérez, A. and Martins, J. P. (1999), Some issues on ontology integration, *in* 'Proceedings of the IJCAI99's Workshop on Ontologies and Problem Solving Methods: Lessons Learned and Future Trends', pp. 7.1–7.12.
Debido a que el término *integración de ontologías* había sido usado en forma indistinta para varias tareas del ámbito de la investigación con ontologías, estos autores proponen discriminar casos en los cuales las ontologías más bien se *mezclan*, se *integran* o se *usan* en lugar de usar simplemente el término se *integran*.

Pinto, H. S. and Martins, J. P. (2001), A methodology for ontology integration, *in* 'K-CAP '01: Proceedings of the 1st international conference on Knowledge capture', ACM Press, New York, NY, USA, pp. 131–138.

Pinto y Martins describen las actividades que componen el proceso de integración de ontologías y describen la metodología para realizar dicho proceso de integración de ontologías. Su metodología está compuesta de los siguientes pasos: (1) identificación de la posibilidad de integración; (2) identificación de los módulos involucrados; (3) identificación de las presuposiciones y los compromisos ontológicos; (4) identificación del conocimiento a ser representado en cada módulo; (5) identificación de las ontologías candidatas; (6) obtención de las ontologías candidatas; (7) estudio y análisis de las ontologías candidatas; (8) elección de las ontologías fuente; (9) aplicación de operaciones de integración, y, (10) análisis de la ontología resultante.

Pollock, J. (1974), 'Knowledge and Justification'.

Pollock, J. (1987), 'Defeasible Reasoning', *Cognitive Science* **11**, 481–518.

Pollock, J. L. (1995), 'Cognitive Carpentry: a blueprint for how to build a person', Bradford/MIT Press.

Poole, D. (1985), On the Comparison of Theories: Preferring the Most Specific Explanation, *in* 'Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI'85)', pp. 144–147.

Poole, D. (1989), 'Explanation and Prediction: an Architecture for Default and Abductive Reasoning', *Computational Intelligence* **5**, 97–110.
Se presenta un marco general para el razonamiento abductivo y el razonamiento default.

Prakken, H. and Sartor, G. (1996), 'A dialectical model of assesing conflicting arguments in legal reasoning', *Artificial Intelligence and Law 4:* pp. 331–368.
Presenta un marco formal para evaluar argumentos en conflicto. El marco tiene la forma de un sistema lógico para argumentación rebatible. Su lenguaje es del estilo de la Programación en Lógica con negación débil y explícita. El sistema admite prioridades, las cuales no son fijas, sino que son derivadas rebatiblemente como conclusiones del sistema.

Prakken, H. and Sartor, G. (2002), The role of logic in computational models of legal argument - a critical survey, *in* A. Kakas and F. Sadri, eds, 'Computational Logic: Logic Programming and Beyond', Springer, pp. 342–380.

Prakken, H. and Vreeswijk, G. (2002), Logics for Defeasible Argumentation, *in* D. Gabbay and F. Guenthner, eds, 'Handbook of Philosophical Logic', Kluwer Academic Publisher, pp. 219–318.
Este survey presenta un análisis de los distintos aspectos más relevantes en argumentación rebatible, así como un análisis comparativo de distintos formalismos.

Raggett, D. (1995), 'Hypertext markup language specification version 3.0'.
Este borrador presenta la especificación del lenguaje HTML versión 3.0.
URL: *http://www.w3.org/MarkUp/html3/html3.txt*

Raggett, D. (1997), 'Client-side Scripting and HTML', W3C Working Draft 14-Mar-97.
El lenguaje HTML (Raggett et al., 1999) es un simple lenguaje de marcado para crear documentos enlazados por hipervínculos que son portables entre distintas plataformas. Esta especificación extiende HTML para soportar guiones ejecutables localmente incluyendo JavaScript, VBScript y otros lenguajes de guiones.

Raggett, D., Hors, A. L. and Jacobs, I. (1999), 'HTML 4.01 Specification. W3C Recommendation 24 December 1999'.
Esta especificación define el Lenguaje de Marcado de Hipertextos (en inglés, *HyperText Markup Language* o *HTML*), versión 4.0, el lenguaje de publicación de la World Wide Web. Además de texto, contenido multimedia e hipervínculos, HTML 4.0 soporta lenguajes de guiones, hojas de estilo en cascada y facilidades para crear documentos para usuarios con discapacidades.

Rahwan, I., Ramchurn, S., Jennings, N., McBurney, P., Parsons, S. and Sonenberg, L. (2003), 'Argumentation-based negotiation', *Knowl. Eng. Rev.* **18**(4), 343–375.

Reiter, R. (1980), 'A Logic for Default Reasoning', *Artificial Intelligence* **13**(1), 81–132.

Rescher, N. and Manor, R. (1970), 'On inference from inconsistent premises', *Theory and decision* **1**, 179–219.

RuleML (2005), 'The RuleML Markup Initiative'.
La Iniciativa para el Marcado de Reglas tiene como objetivo la definición de un lenguaje de reglas compartido llamado *RuleML* que permite encadenamiento hacia adelante o hacia atrás de reglas en XML para deducción, reescritura y tareas de transformación inferencial.
URL: *http://www.ruleml.org/*

Schlobach, S. and Cornet, R. (2003), Non-standard reasoning services for the debugging of description logics terminologies, *in* 'Proceedings of IJCAI 2003'.
Presentan un método novedoso para depurar ontologías en Lógicas para la Descripción y lo aplican a la ontología médica DICE.

Schlobach, S. and Huang, Z. (2005), Inconsistent Ontology Diagnosis: Framework and Prototype, Technical report, Vrije Universiteit Amsterdam.
Presentan un marco para el diagnóstico de ontologías inconsistentes. Desarrollaron dos algoritmos para el marco, uno basado en un razonador externo de Lógicas para la Descripción y el otro utiliza un razonador por *tableaux* especializado.

Sierra, C. and Noriega, P. (2002), 'Agent-mediated interaction. From auctions to negotiation and argumentation', *Foundations and Applications of Multi-Agent Systems – In LNCS Series* **2403**.

Silberschatz, A., Korth, H. F. and Sudarshan, S. (1998), *Database System Concepts. Third Edition*, McGraw-Hill.
Presenta los distintos modelos de bases de datos así como las técnicas de modelado de datos en cada uno de ellos.

Simari, G. R. (1989), A Mathematical Treatment of Defeasible Reasoning and its Implementation, PhD thesis, Washington University, Department of Computer Science (Saint Louis, Missouri, EE.UU.).
La tesis doctoral de G. R. Simari presenta como contribución principal un sistema que modela el razonamiento de un agente inteligente, basándose en el uso de argumentos. Se presenta una implementación computacional de dicho sistema, denominada **justification finder**, implementada en lenguaje Prolog.

Simari, G. R. and Loui, R. P. (1992), 'A Mathematical Treatment of Defeasible Reasoning and its Implementation', *Artificial Intelligence* **53**, 125–157.
Este artículo sintetiza las principales características del formalismo MTDR, presentado originalmente por G.R. Simari en su tesis doctoral. Se incluyen distintos ejemplos que muestran el comportamiento del sistema.

Smirnov, A., Pashkin, M., Chilov, N. and Levashova, T. (2005), Ontology-driven intelligent decision support of OOTW operations: health service logistics support, pp. 522–527.

Smith, M. K., Welty, C. and McGuinness, D. L. (2004), 'OWL Web Ontology Language Guide. W3C Recommendation 10 February 2004'.
Presenta un detalle de todos los constructores del lenguaje de representación de la Web Semántica OWL.
URL: *http://www.w3.org/TR/owl-guide/*

Sterling, L. and Shapiro, E. (1994), *The Art of Prolog. Second Edition.*, The MIT Press.
Presenta el lenguaje de programación Prolog que implementa el paradigma de programación en lógica. Además, presenta las principales técnicas de programación en el mismo paradigma.

Subrahmanian, V. S. and Amgoud, L. (2007), A general framework for reasoning about inconsistency, *in* '20th International Joint Conference on Artificial Intelligence (IJCAI'2007)', Hyderabad, India, pp. 6–12.
Proponen un marco general para el razonamiento con bases de conocimiento inconsistentes en una variedad de lógicas. Modifican la axiomatización de la lógica de Tarski y Scott

desechando los requerimientos de monotonicidad. Para tal lógica definen el concepto de *opción*. Una opción es un conjunto de fórmulas cerrado bajo consecuencia lógica y consistente. Muestran cómo definiendo un relación de preferencia apropiada entre opciones pueden capturar varios trabajos existentes como por ejemplo las subteorías de Brewka (Brewka et al., 1997). Proponen algoritmos para computar las opciones más preferidas.

SWI-Prolog (n.d.).
Brinda un ambiente de programación Prolog bajo la licencia GNU.
URL: *http://www.swi-prolog.org*

Tempich, C., Pinto, H. S., Sure, Y. and Staab, S. (2005), An Argumentation Ontology for Distributed, Loosely-controlled and Evolving Engineering Processes of Ontologies (DILIGENT)., *in* 'ESWC', pp. 241–256.
Se discute un sistema llamado *DILIGENT* para realizar mezcla de ontologías en forma social, *i.e.*por medio de personas discutiendo cuál es la manera más adecuada de modelar un dominio. El sistema permite seguir los hilos de las discusiones señalando los puntos de inconsistencia.

Tešanović, A. (2001), 'What is a pattern?'.
Se discute la problemática de los patrones en el área de la ingeniería de software.
URL: *http://www.ida.liu.se/ uweas/Lectures/DesignPatterns01/tesanovic-WhatIsAPattern.pdf*

van Diggelen, J. (2007), Achieving Semantic Interoperability in Multi-agent Systems: A Dialogue-based Approach, PhD thesis, Universiteit Utrecht.
Se ataca el problema de la interoperabilidad entre agentes en el contexto de un sistema multiagente. Argumenta que una ontología es usualmente insuficiente para proveer la interoperabilidad semántica entre agentes; En cambio, cada agente conoce solamente su propia ontología y no conoce la de los otros agentes. Su acercamiento, basado en diálogos, propone aumentar las ontologías locales de cada agente sólo con la información que necesitan saber para poder interactuar con los otros agentes del MAS.

van Harmelen, F. (2006), 'Semantic Web Research Anno 2006: Main Streams, Popular Fallacies, Current Status and Future Challenges', *Eds.): CIA 2006, LNAI 4149* pp. 1–7.
Se analizan las interpretaciones de la Web Semántica y se discuten los problemas abiertos para la misma.

Verheij, B. (2005), *Virtual Arguments. On the Design of Argument Assistants for Lawyers and Other Arguers*, Asser Press, The Hague.

Volz, R. (2004), Web Ontology Reasoning with Logic Databases, PhD thesis, Universität Fridericiana zu Karlsruhe.
Presenta un análisis de la representación de ontologías expresadas en Lógicas para la Descripción (DL) en la Programación en Lógica (LP); determina qué porción de las DL son representables en LP; realiza un análisis estadístico sobre qué porción de las ontologías encontradas en los ejemplos de la literatura es posible representar con su enfoque.

Wagner, G. (2003), Web Rules Need Two Kinds of Negation, *in* N. H. F. Bry and J. Maluszynski, eds, 'Principles and Practice of Semantic Web Reasoning. Proc. of the 1st International Workshop, PPSW3 '03. Springer-Verlag LNCS 2901'.
Se discute la necesidad de incluir la negación clásica y la negación *default* en los lenguajes de representación de conocimiento para la Web Semántica.

Wang, H., Horridge, M., Rector, A., Drummond, N. and Seidenberg, J. (2005), Debugging owl-dl ontologies: A heuristic approach, *in* 'ISWC 2005, LNCS 3729', Springer, pp. 745–757.
Mucha gente encuentra difícil crear y usar ontologías OWL. Esto ocurre porque una de las mayores dificultades consiste en "depurar" las ontologías: descubrir por qué un razonador ha inferido que una clase es "insatisfacible" (inconsistente). Aún para gente que entiende OWL y el significado de la Lógica para la Descripción subyacente, descubrir por qué los conceptos son insatisfacibles puede ser difícil. Muchos de los razonadores por *tableaux* modernos no proveen una explicación del porqué una clase es insatisfacible. Este artículo presenta un acercamiento heurístico de "caja negra" basado en la identificación de errores comunes e inferencias.

Weber, J. (1997), *Special Edition Using Java 1.1 Third Edition*, Que Corporation.
Presenta una de las versiones primitivas del lenguaje de programación Java.

Wiesman, F. and Roos, N. (2004), Domain independent learning of ontology mappings, *in* 'AAMAS'04'.
Wiesman y Roos proponen un método independiente del dominio para manejar problemas de interoperabilidad por medio del aprendizaje de un mapeo entre ontologías. El método de aprendizaje está basado en el intercambio de instancias de conceptos que están definidos en las ontologías. EL método comienza por identificar pares de instancias de conceptos denotando la misma entidad en el mundo usando técnicas de recuperación de información, seguido por la propuesta y evaluación de mapeos entre las ontologías usado los pares de instancias. Para cada etapa de este método, la probabilidad de que una decisión sea correcta es tomada en cuenta. De acuerdo a los autores, los beneficios del método son: (a) no se requiere ningún conocimiento del dominio, y (b) las estructuras de las ontologías entre las que se debe establecer el mapeo no juegan ningún rol.

Williams, M. and Hunter, A. (2007), 'Harnessing ontologies for argument-based decision-making in breast cancer', *Proc. of the Intl. Conf. on Tools with AI (ICTAI'07)* pp. 254–261.

Introducen el *Marco Argumentativo basado en Ontologías* (en inglés, *Ontology-based Argumentation Framework* u OAF) que enlaza un formalismo lógico basado en argumentación con ontologías expresadas en Lógicas para la Descripción. Muestran cómo estos dos formalismos pueden ser acoplados al observar unas pocas restricciones y proveen características no brindadas por ninguno de los formalismos por separado.

Woolridge, M. (2002), *An Introduction to MultiAgent Systems*, John Wiley & Sons.

Brinda una introducción a los temas principales de la teoría y práctica de los agentes inteligentes y sistemas multiagentes.

Wu, J.-H., Doong, H.-S., Lee, C.-C., Hsia, T.-C. and Liang, T.-P. (2004), 'A methodology for designing form-based decision support systems', *Decision Support Systems* (36), 313–335.

Los sistemas de soporte a la toma de decisiones son tipos especiales de sistemas de información que utilizan formularios para presentar información y para la toma de decisiones. Se utilizan para diseminar información en ambientes de oficina y son útiles para sistemas orientados al usuario. El artículo presenta una metodología que utiliza factorización y síntesis para procesar el conocimiento involucrado en formularios.

Zhang, P., Sun, J. and Chen, H. (2005), 'Frame-based argumentation for group decision task generation and identification', *Decision Support Systems* **39**, 643–659.