ONTOLOGY DATABASES

by

PAEA JEAN-FRANCOIS LEPENDU

A DISSERTATION

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

March 2010

University of Oregon Graduate School

Confirmation of Approval and Acceptance of Dissertation prepared by:

Paea Le Pendu

Title:

"Ontology Databases"

This dissertation has been accepted and approved in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer & Information Science by:

Dejing Dou, Chairperson, Computer & Information Science
Zena Ariola, Member, Computer & Information Science
Christopher Wilson, Member, Computer & Information Science
Monte Westerfield, Outside Member, Biology

and Richard Linton, Vice President for Research and Graduate Studies/Dean of the Graduate School for the University of Oregon.

March 20, 2010

Original approval signatures are on file with the Graduate School and the University of Oregon Libraries.

An Abstract of the Dissertation of

Paea Jean-Francois LePendu for the degree of Doctor of Philosophy

in the Department of Computer and Information Science

to be taken March 2010

Title: ONTOLOGY DATABASES

Approved: _____

Dr. Dejing Dou, Chair

On the one hand, ontologies provide a means of formally specifying complex descriptions and relationships about information in a way that is expressive yet amenable to automated processing and reasoning. When data are annotated using terms from an ontology, the instances inhere in formal semantics. Compared to an ontology, which may have as few as a dozen or as many as tens of thousands of terms, the annotated instances for the ontology are often several orders of magnitude larger, from millions to possibly trillions of instances. Unfortunately, existing reasoning techniques cannot scale to these sizes.

On the other hand, relational database management systems provide mechanisms for storing, retrieving, and maintaining the integrity of large amounts of data. Relational database management systems are well known for scaling to extremely large sizes of data, some claiming to manage over a quadrillion data.

This dissertation defines *ontology databases* as a mapping from ontologies to relational databases in order to combine the expressiveness of ontologies with the scalability of relational databases. This mapping is *sound* and, under certain conditions, *complete*. That is, the database behaves like a knowledge base which is faithful to the semantics of a given ontology. What distinguishes this work is the treatment of the relational database management system as an active reasoning component rather than as a passive storage and retrieval system.

The main contributions this dissertation will highlight include: (i) the theory and implementation particulars for mapping ontologies to databases, (ii) subsumption based reasoning, (iii) inconsistency detection, (iv) scalability studies, and (v) information integration (specifically, information exchange). This work is novel because it is the first attempt to embed a logical reasoning system, specified by a Semantic Web ontology, into a plain relational database management system using active database technologies. This work also introduces the *not-gadget*, which relaxes the *closed-world assumption* and increases the expressive power of the logical system without significant cost. This work also demonstrates how to deploy the same framework as an information integration system for data exchange scenarios, which is an important step toward semantic information integration over distributed data repositories.

CURRICULUM VITAE

NAME OF AUTHOR: Paea Jean-Francois LePendu

PLACE OF BIRTH: Honolulu, Hawaii

DATE OF BIRTH: Aug 29, 1974

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, Oregon
Whitman College, Walla Walla, Washington

DEGREES AWARDED:

Doctor of Philosophy, University of Oregon, Computer and Information Science, 2010
Master of Science, University of Oregon, Computer and Information Science, 2006
Bachelor of Arts, Whitman College, Mathematics, *summa cum laude*, 1996

AREAS OF SPECIAL INTEREST:

Artificial Intelligence
Database Theory
Conceptual Modeling
Computational Logic
Information Integration

PROFESSIONAL EXPERIENCE:

Research & Teaching Assistant, University of Oregon CIS Department
Software Engineer, DB2 Database Kernel and Infrastructure Team, IBM
Programmer / Analyst, Whitman College
Software Specialist, Engineer & Analyst, Harland Financial Solutions
Technical Specialist, Cabletron Systems

GRANTS, AWARDS AND HONORS:

Graduate Teaching Fellow of the Year, University of Oregon CIS Department
National Physical Science Consortium Fellowship
Upsilon Pi Epsilon Honor Society for the Computing Sciences

PUBLICATIONS:

P. LePendu, D. Dou, D. Howe. Detecting Inconsistencies in the Gene Ontology using Ontology Databases with Not-gadgets. in *Proceedings of the International Conference on Ontologies, Databases and Applications of Semantics (ODBASE'09)*, pp. 948–965, 2009.

G, Frishkoff, P. LePendu, R. Frank, H. Liu and D. Dou. Development of Neural Electromagnetic Ontologies (NEMO): Ontology-based Tools for Representation and Integration of Event-related Brain Potentials. in *Proceedings of the International Conference on Biomedical Ontology (ICBO'09)*, pp. 31–34, 2009.

P. LePendu, D. Dou, G.A. Friskoff and J. Rong. Ontology Database: a New Method for Semantic Modeling and an Application to Brainwave Data. in *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM'08)*, pp. 313–330, 2008.

H. Qin, D. Dou and P. LePendu. Discovering Executable Semantic Mappings Between Ontologies. in *Proceedings of the International Conference on Ontologies, Databases and Applications of SEmantics (ODBASE'07)*, pp. 832–849, 2007.

D. Dou, J.Z. Pan, H. Qin and P. LePendu. Towards Populating and Querying the Semantic Web. in *Proceedings of the International Workshop on Scalable Semantic Web Knowledge Base Systems*, pp. 129–142, 2006.

D. Dou, P. LePendu, S. Kim and P. Qi. Integrating Databases into the Semantic Web through an Ontology-based Framework. in *Proceedings of the International workshop on Semantic Web and Databases*, pp. 54, 2006.

D. Dou and P. LePendu. Ontology-based Integration for Relational Databases. in *Proceedings of the ACM Symposium on Applied Computing*, pp. 461–466, 2006.

ACKNOWLEDGMENTS

I would like to foremost thank my parents, family and friends. I also sincerely thank the Computer and Information Science (CIS) faculty, staff and my committee members for their time, encouragement, instruction and support. I have fond remembrances of my assistantships on the Zebrafish Information Network (ZFIN) and Neural ElectroMagnetic Ontologies (NEMO) projects and I thank those colleagues for wonderful experiences. Toward my many advisors I feel the deepest gratitude and respect. In particular, I acknowledge the tireless support of my research and dissertation advisor, Dr. Dejing Dou.

I humbly acknowledge the funding support of the CIS department, Dr. Dejing Dou, ZFIN, NEMO and the National Physical Science Consortium during my studies.

DEDICATION

Dedicated to Jed Schwendiman.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Semantic Web ontologies provide a means of formally specifying complex descriptions and relationships about information in a way that is expressive yet amenable to automated processing and reasoning. As we experience an explosive growth of data being annotated using ontologies, particularly in the biomedical and scientific communities, we strive for the promise of facilitated information sharing, data fusion and exchange among many, distributed and possibly heterogeneous data sources. From the field of knowledge engineering, this fluid, malleable, interconnection of data is one of the major aims of the Semantic Web which has gained significant popularity in recent years. One important challenge for realizing this vision is to develop new reasoning technologies that can scale to handle extremely large and growing data sets. Thus, we define an *ontology database* as a mapping from ontologies to relational database management systems for the purpose of combining the expressive power of ontologies with the scalability of databases.

Recent studies on the scalability of knowledge base systems for the Semantic Web have shown that most demonstrate significant signs of trouble at around one million data instances and completely break down at about three million. For example, the Gene Ontology, already has over 40 million annotated instances as of this writing and it is rapidly growing. Most recent attempts at using databases to address the scalability problem treat the database as a storage and retrieval mechanism for reasoning engines. These approaches take advantage of database features such as persistence, access methods, optimization, and indexing. Our approach, on the other hand, goes beyond that to also consider the database as an active component of the reasoning system itself.

For less expressive logics an ontology database can perform all the necessary reasoning, but when the ontology employs more sophisticated constructs, the database may not be able to guarantee that it will find all possible logical conclusions. In other words, while we can guarantee that an ontology database is always *sound* (that any conclusion derived is correct) it may fail to be *complete* for some ontologies (that not all possible conclusions can be found).

While it has been popular for Semantic Web researchers to assume completeness as a requirement, we assume that a Semantic Web system should guarantee soundness but not necessarily completeness. Indeed, there appear to be some emerging trends toward semi-complete systems in the community. The difficulty is to find balance between expressiveness, efficiency and completeness. In

terms of efficiency, reasoning difficulty generally grows exponentially over the number of concepts in an ontology, making reasoning hard with respect to the ontology. However, if we allow more expressiveness, such as *disjunctions* (e.g., $\phi \vee \psi$) and *case analysis*, then reasoning becomes exponentially difficult over the size of the data, which can be several orders of magnitude larger than the ontology! Some of the best reasoning systems, which have been optimized to handle some of our largest biomedical ontologies being on the order of tens of thousands of terms, are easily crippled in the face of the corresponding data being on the order of tens of millions of instances. On the other hand, if we simply disallow disjunctions altogether merely to guarantee completeness, which has been the approach of late, then we cannot express something as simple as, "He loves me or he loves me not."

One of the goals of our research has been to understand more deeply the relationship between efficiency, expressiveness, and completeness with respect to ontologies and databases. In developing ontology databases, the strategy has been to separate computations that a database can perform from inferences a reasoner should perform. That is, we aim to offload aspects of inferencing to the database management system. What aspects are left over? What can the database *not* compute? If we can separate those inferences that the database cannot perform, then we have identified an area of focus for automated theorem proving.

When using a relational database to assist with scalability issues for knowledge base systems, we carefully consider structural representation as well as

reasoning tasks. In terms of structure, a poorly designed database schema will introduce the problem of having to locate disorganized information. If predicates from the ontology were stored in arbitrarily chosen tables, we would need to perform complex query rewriting to unwind those choices. This not only makes implementation more difficult, but it also introduces an additional layer of potentially inefficient computations. Our systems uses the *decomposition storage model*, which is one of three generalized structural models which bypass the rewriting problem. The decomposition storage model has recently been shown to be more efficient on average than the popular *vertical model* while maintaining many of its desirable features. In terms of reasoning tasks, few approaches besides ours consider in combination the features of a relational database that can make them powerful reasoning systems for the Semantic Web. One of our main contributions has been to explore some of these features of the database.

Similar to the way in which *object oriented databases* were developed to support data management for object oriented programming paradigms, ontology databases are intended to support data management for reasoning on the Semantic Web. What are the performance advantages of this method? What kinds of reasoning can a relational database perform? What kinds of reasoning cannot be performed? Can we extend our methods to support the distributed, heterogeneous and interconnected vision of the Semantic Web?

The remainder of this dissertation is organized as follows. First, in Chapter II we cover the necessary background and related works for understanding the main contributions of our research. Topics range from ontologies and the Semantic Web to data base and knowledge base systems. Next, our main contributions are presented in Chapters III, IV, V, and VI. For each contribution, we include a discussion of our implementations and case studies. The mapping from ontologies to databases is presented in Chapter III. The mapping covers the main features of Semantic Web ontologies, one at a time, and shows how to embed them inside a relational database management system. The two main reasoning tasks, instance checking and consistency, are covered in Chapter IV and V. Chapter VI demonstrates how to use ontology databases to perform information integration, which has been one of our main motivating applications. Finally, we conclude with a general discussion, summary and notes on future work in Chapter VII.

CHAPTER II

BACKGROUND

This chapter covers the background and related works for further understanding the ideas introduced. Topic areas range widely from ontologies and knowledge bases, to deductive and active databases, to information integration and data exchange. We conclude this chapter with an overall summary and problem statement.

Ontologies

The term *ontology* and its related discipline has a long history, but we adopt the definition given by Nicola Guarino in 1998. Taxonomies or categorizations date back as early as Aristotle in the middle of the third century B.C. But the first references to ontologies have been attributed to other philosophers in early 1600 A.D. denoting the science of what is, of the kinds of entities, objects, relationships, processes and other things that exist in reality [67]. Even more recently, ontologies have been adopted by information science researchers to mean a formal specification

of a conceptualization as defined by Tom Gruber in 1993 [37]. An ontology will often specify a set of terms and a set of relationships together with a set of axioms that constrain their possible interactions by using a formal language based on first-order logic. Nicola Guarino [38] further refined Gruber's definition of an ontology as follows:

Definition 2.0.1 (Ontology). *An ontology is an approximation of a conceptualization which is achieved by constraining the possible models of a logical language according to only those intended models (i.e., the ontological commitment) of a given conceptualization.*

The key observation for Guarino's definition is the fact that the intended models of a language may be compatible over many possible worlds of a conceptualization, not just one world in particular. This results in an ontology being merely an approximation of a conceptualization because it does not narrow the meaning of the vocabulary to a single world. What the language does, in other words, is to eliminate those worlds which might be considered absurd. Furthermore, the intended models are not enough to reconstruct the ontological commitment of the language. Therefore, Guarino concludes that the ontology for a language merely approximates a conceptualization if there exists an ontological commitment such that all the intended models of the language are included in the models of the ontology.

Ontologies have become popular in biomedical informatics for standardizing the vocabulary of a domain. However, ontologies also provide the knowledge model for automated reasoning systems, which are referred to as knowledge base systems.

Knowledge Base Systems

A knowledge base (KB) is a logical system in which Γ is a set of formulae which constitutes the knowledge in the system. A KB furthermore provides an inference or proof system for deriving new formulae. The main services a KB offers is the ASK-TELL interface [53] for asserting knowledge and for querying the KB. First, please read "$\Gamma \vdash \phi$" as "Γ *derives* ϕ," and read "$\Gamma \models \phi$" as "Γ *entails* ϕ." If the formulae in Γ contain free variables, then we say that the formula ϕ is entailed by Γ if it is true under all possible variable assignments (i.e., interpretations) in Γ. Another way to see it is that Γ entails ϕ if ϕ is true in all possible models of Γ. We call the proof system of Γ *sound* if $\Gamma \vdash \phi$ implies $\Gamma \models \phi$. If $\Gamma \models \phi$ implies $\Gamma \vdash \phi$, then we call the proof system *complete*. Intuitively, in terms of a query system, soundness tells us that the answers a system can find are correct; completeness tells us the system is capable of finding all possible answers.

Two well-known proof procedures are *modus ponens* and resolution. Resolution is both sound and complete for first-order classical logic, but *modus ponens* is merely sound. If we restrict our logic to Horn Logic, then *modus ponens* is both sound and complete. Horn Logic is a fragment of first-order logic which only

permits formulae with at most one positive literal when presented in conjunctive normal form:

$$\neg\phi_1 \vee \neg\phi_2 \vee \ldots \vee \neg\phi_n \vee \psi$$

We often equivalently view Horn formulae as rules with conjunctions of positive atoms on the left-hand side and a single atom on the right-hand side of the implication symbol:

$$\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n \rightarrow \psi$$

What makes Horn Logic interesting is that the well-known forward-chaining and backward-chaining algorithms based on *modus ponens* are highly efficient. The main idea of *modus ponens* is to satisfy the left-hand side of a rule in order to conclude the right-hand side:

$$\frac{\phi \rightarrow \psi \quad \phi}{\psi}$$

Generalized Modus Ponens (GMP) extends this inference rule with conjunctions of atomic formulae on the left-hand side of a rule, such that the rule is applied whenever a substitution set (denoted by θ) is found for the variables which simultaneously satisfy the conjuncts. This process is called *unification*, and the variable assignments come from the corresponding literals in each of the ground

terms ϕ_i'. After substituting the literals into ψ (denoted by $SUBST(\theta, \psi)$), GMP derives the desired conclusion:

$$\frac{\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n \rightarrow \psi \quad \phi_1' \wedge \phi_2' \wedge \ldots \wedge \phi_n'}{SUBST(\theta, \psi)}$$

Unlike the procedures used for Horn Logic, resolution operates without restriction on the number of positive literals and relies heavily on proof-by-contradiction ($\phi \wedge \neg\phi$ being unacceptable) to reach a derivation:

$$\frac{\phi \vee \psi \quad \neg\phi}{\psi}$$

Resolution is more powerful than generalized *modus ponens* because it does not operate with the same restriction on the number of positive literals:

$$\frac{\phi_1 \vee \ldots \vee \phi_j \vee \ldots \vee \phi_n \quad \neg\phi_j}{\phi_1 \vee \ldots \vee \phi_{j-1} \vee \phi_{j+1} \vee \ldots \vee \phi_n}$$

where each literal ϕ_i other than the resolvent ϕ_j can be either positive or negative.

Knowledge base systems were designed to address representation and reasoning using formal (logic-based) methods. However, without efficient persistent storage methods, these memory intensive applications could not accommodate the scale of growing information systems. Therefore, in the late 1960s to early 1970s methods for efficient storage and retrieval of information were investigated, leading

to one of the most practical contributions of computer science: the relational model and relational database management systems.

Relational Database Management Systems

In 1970 when E.F. Codd developed the *relational model* [19] to address the problem of *data independence*, the idea of separating the logical view of data from its underlying physical implementation. This allowed application programmers to store and retrieve data in a declarative rather than procedural manner, giving front-end applications a high degree of adaptability as underlying disk storage mechanisms and access paths were optimized, reorganized or otherwise changed. Before this, the network and hierarchical database models (which will not be covered here) required fine-grained manipulation of low-level data structures (such as trees and lists). The well-known building blocks of the relational model include: data types, relations, attributes, and schemata [2].

The idea to use the relational model to address data independence did not really take hold until a decade later in the 1980's, when the International Business Machines corporation (IBM) undertook an important project called System R [16], the first influential relational database management system (RDBMS). System R demonstrated that an RDBMS can effectively compete with an experienced programmer, automatically choosing algorithms and data structures to store and retrieve data efficiently at a low (disk and main memory) level. This was the birth

of the ubiquitous SQL language covered in several well-known texts [2, 29] which has resulted in many commercial RDBMSs such as Oracle, Informix, IBM DB2, Microsoft SQL Server, Sybase, as well as popular open source databases such as MySQL and PostgreSQL.

What makes modern RDBMSs most successful are the features that optimize the physical storage and retrieval of tuples on disk, such as partitions and query optimizers, as well as features that maintain data integrity, such as constraints and triggers. Furthermore, a good RDBMS maintains a catalog of information about the database itself (e.g., the size of relations, max and min values, indexes, etc.) to decide which relational algebra expression will cost the least amount of disk access. Of all the features, the query optimizer is what sets commercial RDBMSs apart.

Some years after Codd's paper, in 1976 Peter Chen introduced a higher level representation called the *entity-relationship model* (ER Model) [17] as a design tool bridging the user's conceptual model of his or her data with the relational model. Hull and King [46] provide a nice survey of several other *semantic data models* as well. In the context of biomedical informatics, ontologies and knowledge bases, one open question is how to effectively design a logical database model based on an ontology as a conceptual model. Important strides toward this solving this problem have been made in the fields of deductive and active database systems.

Deductive and Active Databases

There is a rich history of work on the connections between logic, knowledge bases and databases [54], but it was Reiter's seminal works in particular which laid firm groundwork for much of the subsequent research in ontologies, databases and knowledge bases today, including Datalog [69]. Reiter coined the closed-world assumption (CWA) and envisioned databases and inference engines working in concert [60], he reformulated relational database theory in terms of first order logic [62], and he pointed out that the semantics of integrity constraints requires a modal operator [63, 64]. Perhaps as a sign of the times, one of the major assumptions Reiter made is that space is limited and he therefore balanced computation against space.

A deductive database is simply a knowledge base, typically restricted to Horn Logic. Datalog is the famous deductive database system which introduces the use of views and was inspired by the logic programming language, Prolog. A Datalog view has a head and a body. The head is an atomic formula while the body is a conjunction of atomic formulae. Although views correspond directly to Horn formulae, they are often written in a specific syntax, like a Horn rule written backward:

$$\psi \text{ :- } \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$$

What differentiates Datalog from Prolog is that each conjunct in the body may have additional modifiers (e.g., cardinality constraints). Furthermore, popular implementations of Datalog use forward and backward chaining algorithms, while Prolog implementations often use linear resolution. Because of the kinds of modifiers that Datalog allows on conjuncts, including arithmetic operations, Datalog algorithms become undecidable (i.e., they may not terminate) unless additional finite model conditions are added. Datalog provided a crucial step toward thinking of relational databases as logical systems [54], which Reiter helped formalize in [62]. The main difference between database views and Datalog views is that databases do not allow cycles. Therefore, Datalog offers more powerful semantics than traditional database views, using fixed-point models to help guarantee termination. Almost every relational database management system today implements views.

In subsequent work, database researchers found the need to consider alternative approaches for rule-oriented processing within the relational database management system. This lead to the event-condition-action (ECA) paradigm in *active databases* [9, 30, 58]. An ECA rule is commonly referred to as a *database trigger*. The basic idea behind a trigger is to register with the management system a condition that is checked whenever a particular event occurs on a table which transitions the database to a new state. If the condition is satisfied, then the specified action is performed. In general, the action can be any procedure that can vary from possibly taking the database to yet another state, to causing a completely

external process to occur, such as sending an email alert. In a database system, there are only three events that can occur on a table: inserting a record, deleting a record, and updating a record. In the context of logical systems, research has explored the use of triggers for managing constraints [12], for maintaining logical consistency as in a knowledge base [15], for deductive object-oriented databases [22], for maintaining materialized views [13], and some have even gone so far as to consider information integration scenarios [14]. Active database technologies are also common features of relational database management systems today.

Therefore, databases not only provide important data independence features such as query optimizers, data partitioning, and integrity constraints, but they also provide views and triggers. The main question is how to apply the key features of databases toward addressing scalable reasoning over ontologies for the Semantic Web.

The Semantic Web

The Semantic Web is an idea envisioned by Tim Burners Lee *et al.* [5] that brings together all of the fields mentioned above. The Semantic Web is an extension of the World Wide Web (or simply the Web) in which data becomes linked such that machines, not just humans, can automatically combine and manipulate them. The eXtensible Markup Language (XML) was an important leap forward toward providing the structure for marking up data on the Web. However, XML still failed

to provide the key ingredient for combining and manipulating the data as envisioned, which is formal semantics. The World Wide Web Consortium (W3C) has since converged on the Web Ontology Language (OWL) [45] as the defacto standard language for encoding knowledge for the Semantic Web. OWL is a logical language based on Description Logic (DL) [4] for giving data formal semantics using ontologies.

A system based on Description Logic therefore provides the capability to setup a knowledge base, which includes the ability to reason over its contents. DLs divide the knowledge base into two components, the TBox and the ABox. The TBox introduces general knowledge in the form of the terminology, whereas the ABox contains the specific assertions about named individuals using the TBox vocabulary [4]. The two common reasoning tasks for the TBox are *satisfiability* (i.e., descriptions are non-contradictory) and *subsumption* (i.e., whether one term is contained by another). In fact, satisfiability can be reduced to subsumption by asking if any description is subsumed by the empty concept (i.e., the concept which contains no instances). For the ABox, *consistency* (i.e., whether the set of assertions has a model) and *instance checking* (i.e., whether a particular individual is an instance of a give term) are the two main tasks.

OWL extends the Resource Description Framework (RDF), which is an XML language designed for specifying the semantics of data on the Web. RDF allows the specification of classes and properties such that data can be expressed in a very

general subject-property-object graph structure. In addition, RDF Schema provides additional constructs for specifying class and property hierarchies. Furthermore, domain and range restrictions can be applied to properties using RDF. OWL adds additional features such as cardinality restrictions, property characteristics such as transitivity, and additional kinds of property restrictions.

Although every DL statement can be expressed in standard first-order logic, some claim that Description Logic is best suited for the Semantic Web because of the ease of expressing terminologies by focusing on their descriptions. As an example, the statement, "Every offspring has at most two biological parents," can be expressed in DL very concisely:

$$\text{Offspring} \sqsubseteq (\leq 2\ \text{hasBiologicalParent})$$

In contrast, the equivalent first-order logic syntax is much more verbose:

$$\begin{aligned} \forall w, x, y, z : \text{Offspring}(w) & \\ \wedge\ & \text{hasBiologicalParent}(w, x) \\ \wedge\ & \text{hasBiologicalParent}(w, y) \\ \wedge\ & \text{hasBiologicalParent}(w, z) \\ \rightarrow\ & \text{sameAs}(x, y)\ \vee\ \text{sameAs}(x, z)\ \vee\ \text{sameAs}(y, z) \end{aligned}$$

However, few ontologies in the biomedical domain use cardinality constraints. With biomedical ontologies presenting the majority of serious applications, some still question the actual suitability of DL in practice. Regardless of the motivation for choosing DL, research in this area has contributed some of the most important results on the tradeoffs between the expressiveness and tractability of various fragments of first-order logic. Of particular relevance are two DL families: $\mathcal{EL}$ [3] and DL-Lite [11].

The $\mathcal{EL}$ family of languages corresponds to the semantics of many biomedical ontologies including SNOMED [7] and the Gene Ontology [35]. $\mathcal{EL}$ does not include cardinality constraints, but rather focuses on the ability to specify term hierarchies, role inclusions, and basic role restrictions. Research on the $\mathcal{EL}$ family has shown that, in practice, most biomedical ontologies use less expressive ontologies. In particular, the logic of these ontologies is tractable – that is, the algorithms proposed to decide whether the individuals of one term are necessarily contained by another term (i.e., *subsumption*) is sound, complete and terminates in polynomial time [3].

DL-Lite [11] makes another important and relevant contribution by examining DL with respect to databases. DL-Lite defines a family of logical languages which closely captures database schema semantics. Importantly, reasoning over DL-Lite has the important property of being computable in polynomial time for both subsumption inferences in the TBox as well as instance checking on the ABox based on conjunctive queries posed in terms of an ontology.

This presents a significant step toward formalizing the problem of using databases to assist ontology-based reasoning systems. DL-Lite is expressive enough to capture schemas up to total participation constraints (also known as complete constraints or covering axioms) which are a kind of disjunction. An example of a complete constraint would be the statement, "All managers are either top managers or area managers (there are no other kinds of managers)." However, the main features of database management systems that DL-Lite exposes are based purely on structural correspondences between the logic and the schema. That means DL-Lite query answering mainly exploits the query optimization and indexing advantages of databases.

Aside from placing theoretical bounds on the logical language used, many other advances have been made toward addressing the scalability of ontology-based query answering by focusing on optimal structures for storing logical predicates. The most important observation is that almost any knowledge can be encoded in the form of triples. The Resource Description Framework (RDF) is part of the language stack that OWL is built upon, and RDF provides the mechanics for specifying knowledge in terms of RDF triples. The *vertical model* is the most naive implementation in which a single database table with three columns stores all the RDF triples. The important advances on this method fall into two main categories. Firstly, because query answering suffers from excessive and costly self-joins over vertical triple stores (e.g., imagine the difficulty of finding all individuals sharing the

same phone number in a phonebook indexed by last name), some attempt to provide custom optimizations [8, 18, 56] (e.g., imagine creating a special index on phone numbers). The problem with custom optimizations is they do not generalize well and often introduce additional computation steps that do not scale. Secondly, others attempt to side-step the costly self-joins by partitioning the vertical model along properties or concepts [1, 20, 44], which includes the very successful *decomposition storage model* strategy.

In addition to these works, other attempts have seriously considered deeper connections between ontologies and databases. In particular, Motik *et al.* [55] offload integrity constraint checking to the database, further bridging the gap. Their work was motivated by the observations made by Raymond Reiter in an important work connecting knowledge bases with modal logic when it comes to constraints [64]. The key idea is that constraints are unlike regular first-order rules in that they specify knowledge about knowledge. That is, integrity constraints are epistemic in nature. Therefore, to properly specify an integrity constraint requires introducing the ***K**-modal operator* into the logic which Motik *et al.* have proposed for OWL. The important contribution of this work was to stretch the ontology and database relationship further, beyond purely structural considerations. Motik *et al.* have essentially shown how to incorporate another important feature of database management systems into the mix: the ability to maintain a degree of consistency by checking for integrity constraints (e.g., foreign key constraints).

The challenge boils down to addressing the impedance mismatch between ontologies and databases, which should also include the rule-oriented and event-driven features. A few notable works have begun to explore these considerations. Curé and Squelbut explored the use of database triggers in the context of ontologies for truth maintenance [21]. Vasilecas and Bugaite developed an algorithm for transforming ontology axioms into rules [70], and Lee and Goodwin employ similar techniques for managing large-scale applications [49].

Of the many applications for ontologies and the Semantic Web, one of its greatest promises is facilitated information sharing. Often lumped together as information integration, data mediation, sharing, exchange and fusion are among the most difficult yet important challenges of the last few decades.

Information Integration, Sharing and Fusion

For several decades now information integration has been, and it continues to be, a challenging area of research in which ontology-based methods have gained some traction [36, 71]. Information integration is defined as the problem of combining data residing at different sources and providing a unified access to these data through a reconciled global view [50].

For relational databases, integration can be decomposed into query translation or data translation. Query translation is typical in scenarios where a mediator takes incoming queries and delegates sub-goals to the underlying

resources. On the other hand, data translation is more common to migration or exchange scenarios, where information is exported from one location to another.

Definition 2.0.2 (Query Translation). *Query translation is the process of extracting data expressed using one schema to answer the query posed over another schema.*

Definition 2.0.3 (Data Translation). *Data translation is the process of transforming data from a repository under one schema to a repository under another schema.*

Because an ontology can capture the semantics of a database schema (as in DL-Lite), integrating databases reduces to translating queries between two ontologies. The way to do this is to merge the two ontologies and then reason over them together. The key idea is to use namespaces to distinguish terms from each repository and thus safely union concepts between two (or more) ontologies using what are called *bridging axioms* to create a *merged ontology.* We then use an inference engine to translate queries and data between the two KBs using the bridging axioms in the merged ontology.

As an example, Figure 1 depicts the general idea for merging two family genealogy ontologies (DRC-ged[1] and BBN-ged[2]). The representations of the genealogy ontologies highlight a difference in the way families are defined [27]. One

[1] `http://orlando.drc.com/daml/Ontology/Genealogy/3.1/Gentology-ont.dam`

[2] `http://www.daml.org/2001/01/gedcom/gedcom.daml`

could argue that BBN-ged is a more flexible definition since families can include same-sex partners.

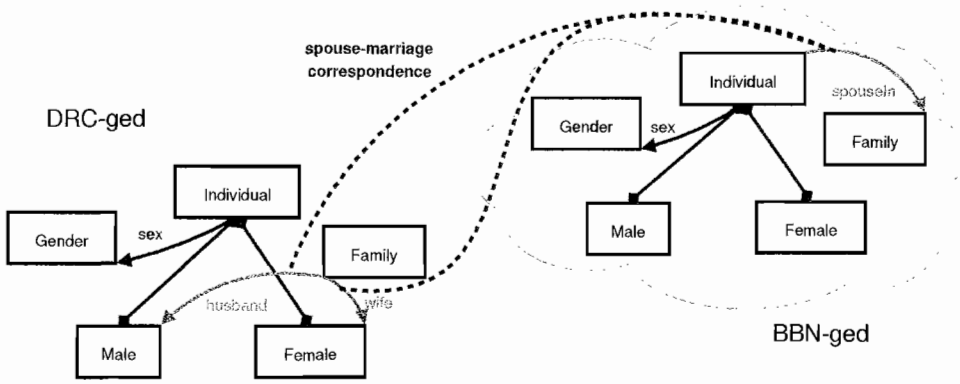


FIGURE 1: Two merged geneology ontologies.

General rules such as the assumption made in DRC-ged families, "for each DRC-ged family, there is at most one husband and wife," are usually encoded in each individual ontology using some first-order logic-based language. Likewise, when ontologies are merged, we include bridging axioms as new first-order rules in the merged ontology with fully qualified namespaces such as the spouse-marriage correspondence from Figure 1 ("husbands and wives in DRC are spouses in BBN"):

$$\forall x, y, z.\ \text{DRC-ged:}Family(z) \wedge Male(x) \wedge Female(y) \wedge husband(z, x) \wedge wife(z, y)$$

$$\Rightarrow \text{BBN-ged:}Family(z) \wedge spouseIn(x, z) \wedge spouseIn(y, z)$$

Ontology-based integration therefore is as simple as making inferences (i.e., "translating") across bridging axioms [25, 26, 27, 28]. Applying this technique to

database integration requires making the simple observation that database schemas are like simple ontologies. It turns out to be a relatively trivial task to lift a schema definition, which is mainly structural, into an ontology (which we have called a *DB ontology*) by following some simple rules of thumb: *relations become classes*, *attributes become predicates*, and *primary keys become instance identifiers*. Extracted using these principles, the DB ontology for one database can be merged with another such that database integration reduces to ontology translation. The main advantage to this approach is that query translation is highly efficient and depends mainly on the size of the ontology (not the data).

However, data translation does not benefit from this approach because it is data-driven. Therefore, the question becomes: how to develop an approach that scales for data translation? In fact, this question has originally motivated much of our research.

Summary

In summary, Description Logics, the logic underpinning Semantic Web ontologies, divides reasoning into *TBox* reasoning (i.e., reasoning about the concepts in the ontology) and *ABox* reasoning (i.e., reasoning about the instances in an ontology). Unfortunately, although existing techniques for TBox reasoning scale adequately for most real-world ontologies [42], on the order of tens of thousands of concepts, one of the major challenges still to overcome is the scalability of reasoning

over annotated data instances in the ABox, on the order of tens of millions to billions of instances.

One study on the scalability of knowledge base systems for the Semantic Web by Guo *et al.* [39, 40] has shown that many memory-based and disk-based systems demonstrate significant signs of trouble around one million instances and completely fail at around three million. This early study made a strong case for the use of database systems to help Semantic Web knowledge bases scale to large numbers of instances.

Simply stated, *the main problem we aim to address is to reason over a given ontology-based knowledge base having a large number of facts using a plain database management system such as MySQL.* In our solution, we present *ontology databases*, a new methodology which uses several first-order logic features of database management systems in combination to directly support the reasoning process. In addition to capitalizing on integrity constraint checking, our work uniquely explores the use of event-driven, active database technologies (triggers) to perform *modus ponens* over extended Horn-logics in a way that has not been done before, by introducing explicit negations and *not-gadgets* into the ontology database structure. Our approach differs from others by not requiring any external theorem prover to perform reasoning; only a plain database management system is required.

CHAPTER III

ONTOLOGY DATABASES

We call the family of databases that model ontologies and answer ontology-based queries *ontology databases*. The key ideas behind ontology databases are to structure the database using a *decomposition storage model* and to map implication rules to database ECA triggers. Furthermore, domain and range restrictions on properties can be mapped to database integrity constraints. This mapping causes the ontology database to compute the minimal model for KBs based on simple ontologies such as the $\mathcal{EL}$ family of biomedical ontologies, which means the ontology database can not only perform efficient query answering but also correct instance checking over a given ontology. If we extend the mapping further to include explicit negations, then the ontology database can also perform a limited form of inconsistency detection.

This chapter explains how to map an ontology to a relational database management system. We use the extended Sisters-Siblings example to illustrate the basic concepts. In this example, the main idea is that if any two people are sisters,

then we can infer that they must be siblings. Our main contribution is the presentation of the mapping, feature-by-feature, from ontologies to databases. Our novel approach is to use database triggers as a key feature in this mapping.

Structure

Arbitrary database structures result in expensive and complicated query rewriting. Therefore, three generic storage models have been extensively studied in the literature – horizontal, vertical and decomposed. Our system employs the decomposed storage model for the reasons described below.

The horizontal model proposes a single, wide table where each attribute (property) is a column. It is rarely used because it contains excessively many null values (because not every entity participates in every property) and is expensive to restructure because adding new predicates requires new columns.

Conversely, the vertical model presents a single, tall table with only three columns: subject, property and object. It is quite popular because it avoids the two main drawbacks of the horizontal model. Furthermore, the vertical model supports very fast insertion of new data. In fact, Sesame [8], a popular Semantic Web RDF storage framework, and most other RDF triple stores use the vertical storage model. Unfortunately, the vertical storage model is prone to slow query answering performance because of the excessive number of joins against the single, tall table for most queries. Furthermore, type membership queries are somewhat awkward.

The typical workaround is to first partition the vertical table to better support type membership queries, then to partition it further along predicates that will optimize joins along advantageous distributions. However, the partitions often seem arbitrary and lead back toward the query rewriting problem.

The decomposition storage model can be viewed as a fully partitioned vertical storage model, where the single table is completely partitioned along every type and every predicate. That is, each type and each predicate gets its own table. When taken to this extreme, query rewriting becomes straight forward again. The decomposition storage model keeps the advantages of the vertical model while improving query performance. Although it is possible to customize more efficient partitions, the cost of implementing the proper query rewriting tools is seldom worth the benefit [1, 20].

Domain and Range Restrictions

Domain and range restrictions on properties in an ontology correspond to integrity constraints on the subject and object of a property table, which can be implemented by using foreign-key constraints referencing a type table or by assigning the respective datatype (for datatype properties). For example, we might restrict the domain and range of Sisters to Person as in Figure 2, where the dashed lines and checkmarks denote a foreign-key (f-key) check. That is, before the

assertion Sisters(Lily,Zena) is loaded, the database first verifies that Lily and Zena are already in the Person table, otherwise an error is raised.

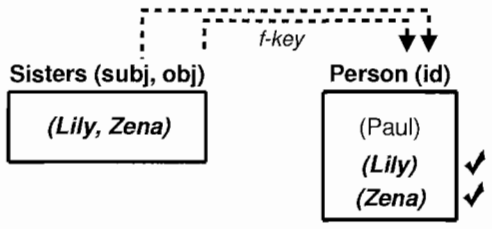


FIGURE 2: Restrictions using integrity constraints.

The domain and range restriction on Sisters object property would be implemented in SQL as follows:

```
CREATE  TABLE sisters (
            subject VARCHAR NOT NULL,
            object VARCHAR NOT NULL,
            CONSTRAINT fk-sisters-subject-person FOREIGN KEY (subject)
              REFERENCES person(id)
              ON DELETE CASCADE,
            CONSTRAINT fk-sisters-object-person FOREIGN KEY (object)
              REFERENCES person(id)
              ON DELETE CASCADE )
```

Semantically, the first order formulae for restrictions go beyond Horn Logic because they require negations. It has also been noted in the literature that we require the **K** modal operator [4, 64] to precisely describe the semantics of these features. The first order logic for the Sisters-Person restriction should technically be the following:

$$\forall x, y : \neg \mathbf{K} Person(y) \rightarrow \neg \mathbf{K} Sisters(x, y)$$

Subsumption

The sub-class and sub-property (*is_a*) axioms define the subsumption hierarchy (*inclusion axioms*) over which we can perform a majority of inferences. In the literature, these have been mapped to views [57]. However, we propose mapping the *is_a* relationship to a database trigger. One important reason for our approach is that few systems support foreign-key checks on views – making it difficult to consider restrictions in combination with subsumption.

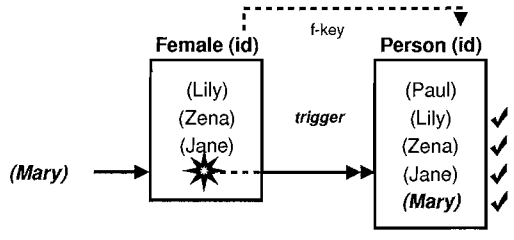


FIGURE 3: Subsumption using triggers.

As an example, if Female is a sub-class of Person, then our system forward propagates instances of Person as instances of Female are asserted by using database triggers. Figure 3 illustrates how asserting Female(Mary) triggers (denoted by the starburst and double arrow) the assertion of Person(Mary). The SQL implementation would be the following:

CREATE TRIGGER subclass-female-person

ON INSERT (x) INTO female

FIRST INSERT (x) INTO person

Semantically, *is_a* relationships correspond to Horn rules, such as the following Female-Person subsumption:

$$\forall x : Female(x) \rightarrow Person(x)$$

However, triggers only capture part of the semantics intended. In particular, the trigger corresponds semantically to the following rule [4]:

$$\forall x : \mathbf{K}Female(x) \rightarrow \mathbf{K}Person(x)$$

To fully capture the intended semantics of the inclusion axiom, we should also consider the contrapositive, which corresponds to a restriction (implemented as an

f-key in Figure 3):

$$\forall x : \neg \mathbf{K} Person(x) \rightarrow \neg \mathbf{K} Female(x)$$

Note that the "FIRST" keyword in the SQL for the trigger ensures that the temporal dependency between the trigger and the integrity check is correctly maintained. It may seem redundant to include a check since the trigger will guarantee existence, however it is necessary to have this check to maintain consistency going forward.

Cardinality

We support only limited cardinality constraints on properties, zero or one, for minimal or maximal participation. For example, the rule, "People have at most one social security number (SSN)," is a maximal cardinality constraint of one on the property hasSSN. The following axioms capture our everyday assumptions about SSNs:

$$\forall x : \neg\mathbf{K}Person(x) \rightarrow \neg\mathbf{K}\exists y : hasSSN(x, y)$$

$$\forall x, y : \neg\mathbf{K}y : String[\text{“}ddd - dd - dddd\text{”}] \rightarrow \neg\mathbf{K}hasSSN(x, y)$$

$$\forall x, y, z : hasSSN(x, z) \wedge hasSSN(y, z) \rightarrow equal(x, y)$$

$$\forall x, y, z : hasSSN(z, x) \wedge hasSSN(z, y) \rightarrow equal(x, y)$$

$$\forall x : \mathbf{K}Person(x) \rightarrow \mathbf{K}\exists y : hasSSN(x, y)$$

The first two rules are restrictions; the second one specifying a datatype property restriction. The third and fourth rules specify that no two people have the same SSN and no person has two SSNs, respectively. The last rule adds that every person has at least one SSN. We use uniqueness constraints (e.g., a primary or alternate key) to implement maximal cardinality. Cardinality axioms that have the same semantics as restrictions are treated as such. For example, for the hasSSN property, we would create the table as follows in SQL:

```
CREATE  TABLE hasSSN (
        subject VARCHAR NOT NULL,
        object VARCHAR (format:"ddd-dd-dddd") DEFAULT NULL,
        UNIQUE (subject),
        UNIQUE (object),
        CONSTRAINT fk-hasSSN-subject-person FOREIGN KEY (subject)
          REFERENCES person(id)
          ON DELETE CASCADE )
```

Minimal cardinality, on the otherhand, is more interesting. As the rule suggests, it should be implemented as a positive trigger rule, but the existential quantifier does not provide a specific value to forward propagate – merely that one exists. As Reiter [64] suggests, we use a *null* value. Therefore, we would add a trigger to complete the minimal cardinality constraint:

```
CREATE TRIGGER  exists-person-hasSSN
        ON INSERT  (x) INTO person
                INSERT  (x,NULL) INTO hasSSN
```

Null values can be difficult to work with in a KB setting. Using them in this way can be considered a weak form of skolemization. What would be more useful is to create a new skolem term, a variable that is functionally dependent on the quantifiers that determine it. V-tables [47] offer an interesting alternative, a more powerful approach that we did not choose because they are not widely adopted in database management systems.

Explicit Negation

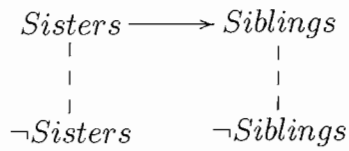
The closed world assumption (CWA) assumes that all facts are false by default until asserted or inferred otherwise [61]. We differentiate explicit negations from those that might be assumed from the CWA. That is, the explicit negation asserting that Bob is *not* Female is treated differently from *Bob* being excluded from the Female table in an ontology database allowing negations. If we allow negations in the ontology database, then for every positive table (e.g., Female), we include a corresponding explicit negation table (e.g., ¬Female) in the database. Furthermore, an *exclusion dependency* enforces the fact that ϕ and $\neg\phi$ cannot be simultaneously true. An exclusion dependency is a special kind of integrity constraint which ensures a tuple does not appear in both a positive and its corresponding negative table at the same time.

If we consider the concept graph with triggers as directed arrows between concept atoms, then the Sisters-Siblings sub-property would look like the following

diagram:

$$\text{Sisters} \longrightarrow \text{Siblings}$$

By adding exclusion dependencies (e.g., the constraint that the same tuple cannot appear in both ϕ and $\neg\phi$) as undirected, dashed arrows, the concept graph takes on a distinctive general structure, as in the following:



We refer to these structures as *not-gadgets*.

Not-gadgets help to detect inconsistencies as well as to answer a limited form of disjunctive query. As an example of the latter, suppose we have a simple ontology which states that, "All Persons are either Female or Male," and its extension, "Bob and Jane are Persons. Jane is a Female, but the person Bob is not a Female." The first order axiom, called a covering axiom, is in the form of a disjunction, but we can transform it into a series of conjunctive statements with negations as follows:

$$\forall x : Person(x) \rightarrow (Female(x) \vee Male(x))$$

$$\forall x : Person(x) \rightarrow (\neg Female(x) \rightarrow Male(x))$$

$$\forall x : (Person(x) \wedge \neg Female(x)) \rightarrow Male(x)$$

$$\forall x : (Person(x) \wedge \neg Male(x)) \rightarrow Female(x)$$

The transformations rely on the following well-known theorems of classical logic, respectively:

$$(\phi \vee \psi) \equiv (\neg\phi \rightarrow \psi)$$

$$(\phi \rightarrow (\phi' \rightarrow \phi'')) \equiv ((\phi \wedge \phi') \rightarrow \phi'')$$

$$(\phi \vee \psi) \equiv (\psi \vee \phi)$$

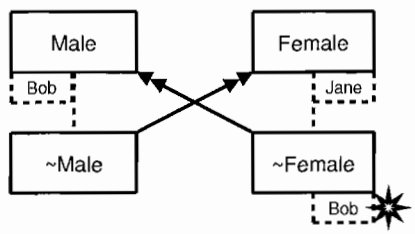


FIGURE 4: The Male-Female not-gadget.

Given just the conjunctive statements, we can still pose the disjunctive query, "Is Bob either a Male or Female?" as a union over the Male and Female tables and expect a concrete answer: true. In fact, the ontology database with negations will explicitly store the fact that Bob is Male as shown in Figure 4. Ordinary relational database modeling techniques would not return the correct answer. Note, however, that without the negative fact Bob is not Female, neither would an ontology database. This is the fundamental limitation of ontology databases – they are not guaranteed to be complete. We believe that this is ultimately due to the constructive nature of relational database management systems. Because an RDBMS concerns itself with concrete knowledge, it can never

be as powerful as full classical logic with the non-constructive inference rules (case analysis and *reductio ad absurdum*).

Disjunctive queries therefore reduce to unions of conjunctive queries. The Male-Female covering axiom would be implemented in SQL no differently than inclusion axioms, this time using explicit negations appropriately:

```
CREATE TRIGGER  disjunction-female-male
ON INSERT       (x) ¬female
FIRST INSERT    (x) INTO male WHERE (x) IN
                (SELECT y FROM person)
```

Summary

In summary, we have mapped four important features of ontologies to database management systems. First, domain and range restrictions map to integrity constraints. Second, *is_a* relationships map to triggers as well as integrity constraints. Third, maximal and minimal cardinality constraints are mapped to integrity constraints and triggers, respectively. Fourth, a limited form of negative and disjunctive knowledge are mapped to not-gadgets with exclusion dependencies. Finally, we use the decomposition storage model to easily store and retrieve instances of terms from the ontology.

Definition 3.0.4 (Ontology Database). *An ontology database is a database that models an ontology for supporting ABox reasoning (instance checking and consistency).*

Figure 5 depicts each of the components described so far working together: structure, restriction, subsumption, negation. For example, the Male-Female covering axiom (bottom-left area of the figure) will propagate Male(Paul) while verifying that ¬Male(Paul) is not the case. Also, the cardinality constraints on hasSSN (middle-right area of figure) will propagate *null* values as depicted. Finally, as the Sisters-Siblings subsumption relationship is maintained (top area of the figure), it is checked against the appropriate domain and range restrictions. In this case, we have further restricted the Sisters domain and range to Female as indicated by the dashed, foreign-keys.

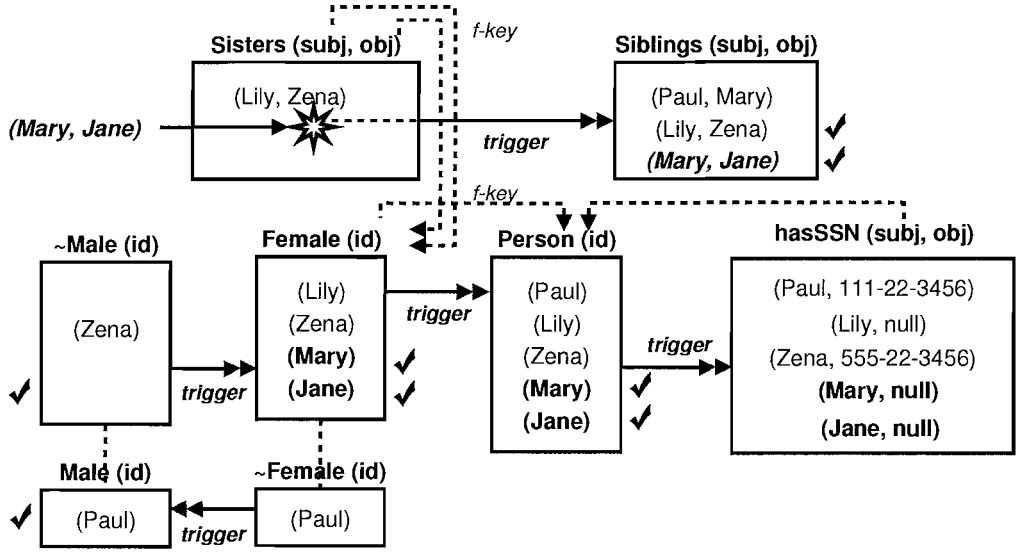


FIGURE 5: The full Sisters-Siblings, Male-Female, hasSSN example.

CHAPTER IV

INSTANCE CHECKING

Query answering is the generalized problem of determining whether a predicate is true, which is called *instance checking* in Description Logic systems. In the context of an ontology and KB system, query answering requires that subsumed instances also be considered in the query answering process. The subsumption hierarchy is primarily defined by the sub-class and sub-property axioms. Therefore, when asking whether an instance belongs to a particular class, it requires asking whether the individual belongs to all of the subsumed classes as well. We can inductively define *instance checking* as follows:

Definition 4.0.5 (Instance Checking). *Instance checking is a reasoning process that returns* true *for $\phi(\alpha)$ whenever either $\phi(\alpha)$ is the case, or $\phi'(\alpha)$ is the case and $\phi' \sqsubseteq \phi$ (similarly for properties or binary relations).*

For Horn Logic, *modus ponens* is both sound and complete. Therefore, since triggers implement a forward chaining algorithm for *modus ponens*, an ontology database can return all of the correct answers to conjunctive queries over ontologies

using features up to Horn Logic, such as *is_a* hierarchies. What differentiates conjunctive queries over ontologies from those over traditional schemas is that answers to each atomic query sub-goal should include instance checking over subsumed terms.

An ontology database answers conjunctive queries in polynomial time with respect to the data because the full extension of each term is pre-computed and we perform no instance checking reasoning at query-time. Therefore, computing answers takes only as long as the corresponding SQL query takes to run on the database.

This chapter presents several case studies and discusses the benefits and tradeoffs of our implementation with respect to conjunctive query answering. Our main contribution is to analyze the performance benefits and tradeoffs of using a trigger-based approach versus other traditional approaches. Also, we propose a new benchmark for KB systems by varying not only the size of the extension, but also the size and complexity of the ontology itself. Finally, we illustrate a novel application domain which has provided promising results on both the usability and performance of our implementation.

Case Study: Query Answering Performance

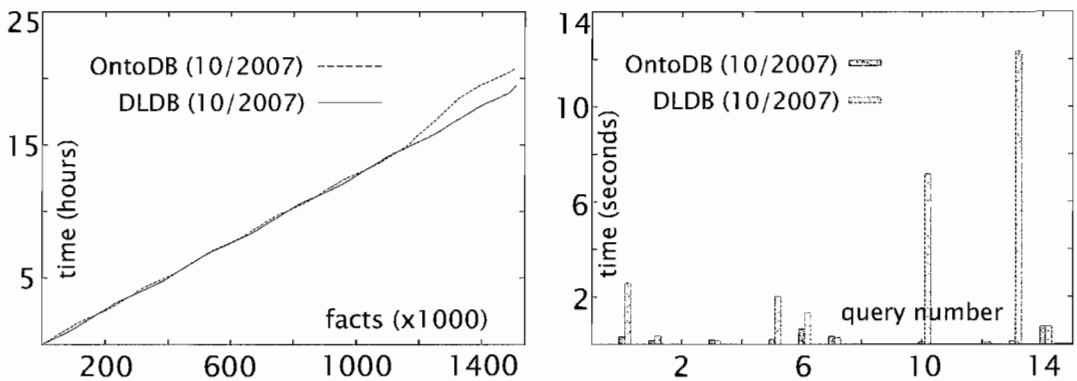
The fundamental difference between using views and triggers is the notion of pre-computing the inference. In the view-based approach (non-materialized), the

query is unfolded and answers are retrieved at query-time. In the trigger-based approach, knowledge is forward-propagated as it is asserted. Clearly, the benefit of forward-propagating knowledge, as with materialized views, is faster query response time. In other words, we amortize the cost of pre-computation over the number of queries, expecting queries to outnumber assertion events over time. Therefore, when measuring the performance of a query answering system, it is important to also consider the time it takes to load all of the assertions.

The Lehigh University Benchmark (LUBM) [41] is a framework for measuring the scalability of a knowledge base system over extensions of varying sizes. LUBM includes the University Ontology, a data extension generation mechanism, and a set of 14 queries. The benchmark requires the system to report both load-time and query answering performance. Query performance times should be averaged over at least 10 trials, and then averaged again over 3 sessions. Between each session, all caches should be flushed. Load-time is the total time it takes to load the ontology as well as the generated data extension into the KB, whether or not it is an in-memory or disk-based system. The LUBM data extension generator takes two parameters which varies the size of the extension. The parameters represent (1) the number of universities, and (2) the number of departments per university.

In [51] we evaluated ontology databases using triggers by comparing it with the DLDB system [40, 39, 57] which is a very similar system using views. This experiment was performed in October of 2007 using an unremarkable laptop having

a 1.8 GHz Centrino processor and 1GB RAM. We used the $\text{LUBM}_{10,20}$ benchmark and experimentally confirmed that by using triggers rather than views, query performance clearly benefits by several orders of magnitude. However, we were surprised to discover that it came at no observable cost in terms of load-time. We expected load-time to increase by a factor proportional to the average depth of the subsumption hierarchy, which for the LUBM is about three.



(a) Load time. (b) Query time.

FIGURE 6: $\text{LUBM}_{10,20}$ performance results.

The overlapping trends in Figure 6(a) illustrates how the load-time for views (DLDB) versus triggers (OntoDB) appears unaffected. In the figure, there is an anomaly at around 1.2 million facts which we later confirmed was explained by disk contention resulting from a virus scanner.

We used a non-logarithmic scale in Figure 6(b) to contrast how dramatically different DLDB and our system, OntoDB, perform, especially on chain queries (queries 10 and 13). Queries 2, 4, 8 and 9 are not shown in the figure as they have extremely long running times showing no significant difference in performance for

both the OntoDB and DLDB. These queries fall into the class of "complete queries" which have many join predicates.

Finally, having stored the full transitive closure of all instances over the subsumption hierarchy, the actual disk-space usage for LUBM using OntoDB roughly triples in comparison to DLDB, as we might expect based on the average depth of the subsumption hierarchy of the University Ontology, which has on average depth three, because the OntoDB system literally copies data up the hierarchy.

The surprising result regarding load-time prompted us to further explore the performance of our OntoDB system along various other parameters that the LUBM does not test, namely the size and depth of the ontology. The LUBM benchmark uses a fixed ontology, the University Ontology, and varies the number of instances for that ontology to compare the scalability and performance of knowledge bases over large numbers of instances. However, the University Ontology is fairly small. There are only about 78 terms in a subsumption hierarchy averaging a depth of three with a maximum depth of five. Compared with biomedical ontologies such as the Gene Ontology[1] (on the order of 30-thousand terms up to depth 18) and SNOMED [7](on the order of 270-thousand terms), the University Ontology is minuscule.

[1] `http://berkeleybop.org/goose`

Case Study: Load-time Performance

Our theory for the surprising lack of visible load-time cost for the LUBM case study is that for shallow hierarchies, the database can parallelize the writes to several tables at once, depending on the number of write-heads on the disk. Hence, for ontologies of significantly larger depth than the LUBM ontology, we would expect the propagation cost to become more evident. Therefore we developed the following experiment in which we randomly generated subsumption hierarchies of varying complexity to better understand the factors that contribute to the load-time performance of our ontology database system with regard to the subsumption hierarchy. This experiment was performed in October of 2009, using an upgraded desktop computer having a 2.2 GHz Dual Core processor and 8GB RAM.

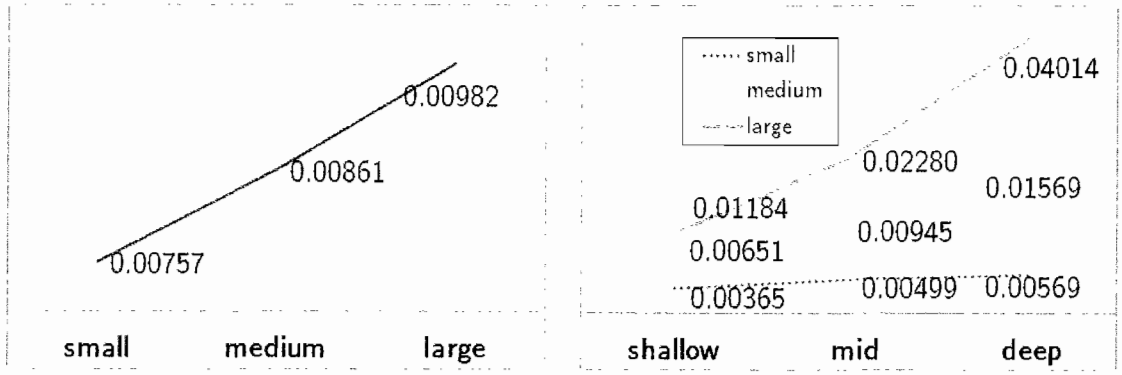
We created some software that takes two parameters as input, depth and size, and generates an ontology subsumption hierarchy having the supplied depth and total number of terms. Size is the total number of terms or nodes in a hierarchy; depth is the maximal path length from the root node to leaves. We note that the software also takes fan-out (number of children) and density (probability of maximal fan-out) parameters to help randomize the hierarchy structure, but these parameters are of little direct interest in this study. Finally, the software generates a target number of extensional data, which are asserted instances of randomly chosen terms in the ontology (occurring at uniformly chosen depths).

	ontology parameters		time to load database schema		time to load data instances	
	size	depth	(means per term)		(means per instance)	
small	78	5	0.00695	0.00757	0.00365	0.00478
	81	10	0.00712		0.00499	
	72	20	0.00864		0.00569	
medium	1623	5	0.00821	0.00861	0.00651	0.01055
	1555	10	0.00893		0.00945	
	1827	20	0.00868		0.01569	
large	19992	5	0.00957	0.00982	0.01184	0.02493
	22588	10	0.00959		0.02280	
	19578	20	0.01028		0.04014	

FIGURE 7: Additional load-time study ontologies.

Figure 7 summarizes the parameters of nine different ontologies we generated and used for our study. In the table are the nine different ontology parameters together with the corresponding schema load time and the data instance load time for each. Times are measured in seconds. Each ontology is classified according to size and then depth. The small ontologies have on the order of 100 terms; medium around 2,000 terms; and large around 20,000 terms. For each ontology size group, we also vary the depth from 5 to 10 to 20 (shallow, mid, deep). As mentioned, for each ontology, we also generate data instances that are randomly scattered throughout the class hierarchy. That is, for each data item, we randomly choose a term from the ontology, regardless of position in the hierarchy, and make that data an instance of the randomly chosen term. We measure (1) the time to load the database schema that our tool generates, and (2) the time to load the data instances into that database.

Our hypotheses were each confirmed: (1) the schema load time will be dependent on the size of the ontology but not dependent on depth (nor fan-out, nor density); (2) within each ontology, the time it takes to load a single instance will be constant, not depending on the total number of instances previously loaded; (3) over all the ontologies, the time it takes to load a single fact will be proportional to the depth and size of the ontology. We ran a standard analysis of variance (ANOVA) test on the resulting metrics to confirm the first hypothesis that size has the only significant effect on the schema load time and that depth, fan-out and density have no significant interactions. Figure 8(a) shows that as the ontology size grows significantly, the time to load the ontology increases. This can be very likely explained by the performance of file systems at the operating-system level because MySQL uses a file-per-table model for schematic information. Most operating system file tables take performance hits as the number of files exceed certain thresholds.



(a) Schema load-time.

(b) Instance load-time.

FIGURE 8: Additional load-time study results.

Our second hypothesis was also confirmed: the time measures after every 1000-th insertion (up to one million) showed a steady, insignificant 3.9% average standard deviation. Finally, our third hypothesis was of the most interest and is summarized in Figure 8(b). The chart shows that both size and depth do play a significant role in the insertion-time of each fact into the ontology database. As we expected, depth shows a steady, near-linear influence on instance load time. It is also clear that the number of terms (respectively, tables) in the ontology (respectively, database) also has a distinct influence on the performance of instance loading. What is not entirely clear, however, is the specific interaction between ontology size and depth. For example, the medium-deep ontology takes more time to load instances than a large-shallow ontology. This would be an interesting area of future study.

Case Study: Neural Electromagnetic Ontologies

The Neural ElectroMagnetic Ontologies (NEMO) [24, 32] project decomposes, classifies, labels, and annotates event related potentials (ERP) data using ontological terms. ERPs are measures of brain electrical activity (EEG or "brainwaves") that are time-locked to experimental events (e.g., the appearance of a word). These measures provide a powerful technique for studying brain function, because they are acquired non-invasively and can therefore be used in a variety of populations – e.g., children and patients, as well as healthy adults. In addition, they

provide detailed information about the time dynamics, as well as the scalp spatial distribution, of neural activity during various cognitive and behavioral tasks.

The NEMO project aims to include an ontology database component that will store large numbers of ERP datasets collected from multiple research sites. The database will support ontology-based querying and reasoning for complex queries such as the following:

> *Return all data instances that belong to ERP pattern classes which have a surface positivity over frontal regions of interest and are earlier than the N400.*

In this query, "frontal region" can be unfolded into constituent parts (e.g., right frontal, left frontal; see Figure 9). At an even more abstract level, the "N400" is a pattern class that is also associated with spatial, temporal, and functional properties.

In Figure 9, (a) is a representation of concepts from the NEMO ERP ontology used for this preliminary case study; P100 pattern and medial-frontocentral (MFRON) channel groups are highlighted; (b) is a 128-channel EEG waveplot with positive voltage plotted up for responses to words versus non-words; (c) is a time course of the P100 pattern factor for same dataset, extracted using Principal Components Analysis; (d) is a topography of the P100 factor with negative on top and positive at bottom; (e) is an international 10-10 layout with electrode location 'Fz' highlighted, which is placed on the

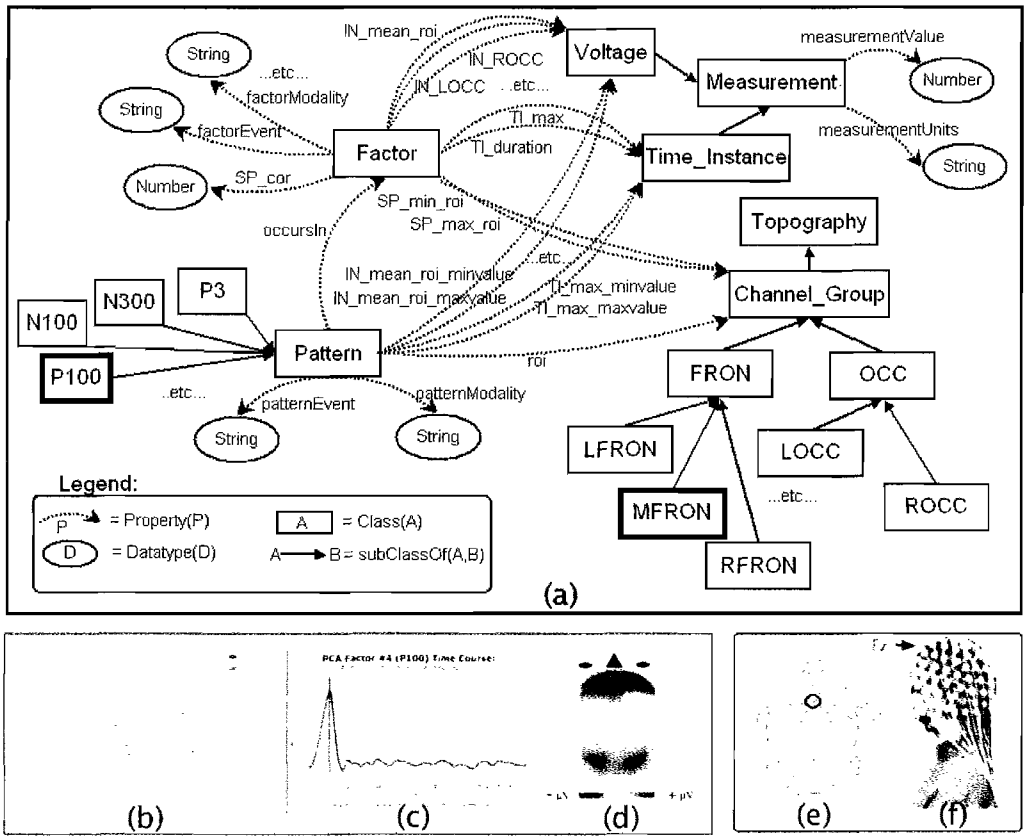


FIGURE 9: Neural Electromagnetic Ontologies (NEMO).

medial-frontocentral scalp region; and (f) is an EEG net applied to the scalp surface of a human subject. See [33] for additional background on ERP data.

Preliminary results on the application of ontology databases for NEMO have been very promising. In particular, the ability to pose queries at the conceptual level, without having to formulate SQL queries that take the complex structural interactions and reasoning aspects into consideration, was very attractive to the neuroscientists. In our preliminary study, we tested several queries similar to the one above that examined ease of formulation, aggregation, subsumption, and total number of instances against data that was annotated using an ontology similar to the one in Figure 9. For example, we measured the time it takes to answer the following queries:

> *Which patterns have a region of interest that is left-occipital and manifests between 220 and 300ms?*
>
> *What is the range of intensity mean for the region of interest for N100?*

In every case, we found it easy to formulate the domain-scientist's queries using terms from the ontology, not having to worry about the subsumed terminology. Furthermore, the system achieved 100% precision and recall, providing exactly the answers expected by our domain experts. This result was not surprising, given that the NEMO ontology used for this study consisted mainly of *is_a* relationships. Finally, the performance for every query was extremely fast, on the order of five to ten milliseconds. Newer iterations of the NEMO ontologies [32] (see Figure 10) will

likely incorporate disjointedness and completeness constraints and will motivate using ontology databases with negations and not-gadgets as a future case study.

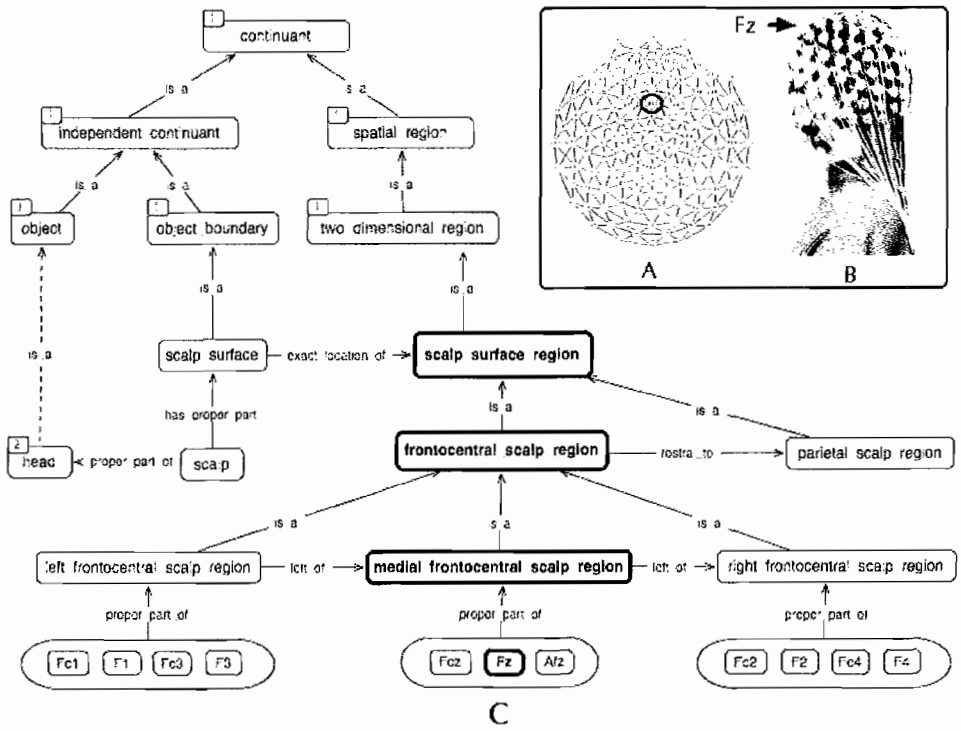


FIGURE 10: A portion of the NEMO spatial ontology.

Discussion

For simple *is_a* hierarchies, an ontology database using triggers is, for all intents and purposes, the same as an ontology database using materialized views. However, it can be difficult to maintain materialized views and some database management systems, such as MySQL, do not even provide this advanced feature. It would be fair to say that an ontology database using triggers appears to be

equivalent to a mechanism that keeps the appropriate materialized views always up-to-date. In that sense, what we have contributed is an approach for reasoning over Semantic Web ontologies that provides the same functionality as materialized views might provide, but one that is more universally supported across underlying relational database management platforms.

Another important observation we would like to make is that ontology databases using triggers is a specialized implementation of the typical forward-chaining algorithm for *modus ponens*:

```
Procedure ForwardChain(KB,fact)
   FOR EACH RULE in the KB in which the
      fact's predicate appears in the premise
   DO
      ADD fact to the KB
      UNIFY the RULE premise against the KB
      APPLY the variable substitution found
         during unification to the RULE's conclusion,
         consider that a new_fact
      ForwardChain(KB,new_fact)
   UNTIL no new facts are generated
```

Two things differentiate our implementation: (1) the KB resides in persistent storage (on disk), (2) the unification phase is efficient (logarithmic time) because it

takes advantage of the intrinsic database indexed search capabilities. These are both important differences for performing reasoning over very large KBs. Firstly, the persistent storage allows the KB to incrementally grow over time. Many KB reasoning systems are memory-based, loading all facts into memory, while performing reasoning over a single session. For large datasets, memory-based systems with either crash or begin to inefficiently thrash against the disk as virtual memory is swapped. To address these problems, caching mechanisms can be employed, but it becomes difficult to decide precisely what and how much data to cache during the unification process. Relational database management systems do this automatically and with proven success. Our implementation takes advantage of this feature. Secondly, after a session terminates, the traditional KB system will have to repeat the process if the same facts (or more) facts want to be reasoned over, but an ontology database system can simply pick up where the last session left off, allowing more facts to be added without having to recompute the previously drawn inferences. In other words, the ontology database amortizes the cost of precomputing and storing the inferences, making them outperform in scenarios where queries are performed with significantly higher frequency than assertions.

CHAPTER V

INCONSISTENCY DETECTION

Consistency is another important reasoning service provided by DL reasoners. ABox consistency is related to TBox satisfiability, which is the problem of ensuring that descriptions are non-contradictory. Whenever a KB with a satisfiable TBox entails that an individual is both an instance of a term as well as its complement, then the KB is inconsistent (contradictory). That is, for those worlds in which that particular ABox holds, the KB is not satisfiable. Therefore, a KB is consistent if for a given ABox there exists some extension for which the KB is satisfiable.

Definition 5.0.6 (Inconsistency Detection). *Inconsistency detection is a reasoning process that returns* true *whenever* $\phi(\alpha)$ *and* $\neg\phi(\alpha)$ *are the case (similarly for properties or binary relations).*

Detecting inconsistency is the dual problem to consistency checking: if inconsistency is detected, then the ABox is not consistent. If no inconsistency is detected we can only claim that the ABox is consistent if the inconsistency detection algorithm is proven to be complete. Because we restrict ourselves to *modus ponens*,

we cannot make that claim because we are required to consider negations which take us beyond Horn Logic. Recall, Horn Logic is complete for *modus ponens* only for rules having no negations. Therefore, although we demonstrate how to perform inconsistency detection using ontology databases, we cannot claim that we are solving the stronger problem of consistency checking for ABox reasoning.

This chapter introduces *not-gadgets* into the ontology database system to handle inconsistency detection. Our main contribution is the definition and implementation of a *not-gadget* and the accompanying *exclusion dependency*. As an interdisciplinary contribution, we also apply this theory to solve an important open problem in biomedical ontology research, which we summarize as a case study.

Not-gadgets

We introduced the concept of not-gadgets in Chapter III which as been reported in [52]. The definition of a not-gadget relies on the notion of an exclusion dependency, and we formally define these terms as follows:

Definition 5.0.7 (Exclusion Dependency). *An exclusion dependency is a special form of integrity constraint which raises an error whenever a tuple is attempted to be inserted into both the positive and the negative tables of a not-gadget. An exclusion dependency enforces the axiom:* $\neg \boldsymbol{K}(\phi \wedge \neg\phi)$.

Definition 5.0.8 (Not-gadget). *A not-gadget adds, for every positive term (respectively, table), a negative counterpart intended to store explicit negations together with an exclusion dependency. In an ontology database using triggers, a not-gadget also extends database triggers implementing any rule of the form $\phi \rightarrow \psi$ to also include rules of the form $\neg\phi \rightarrow \psi$. (As for GMP, the triggers may be similarly extended to include negations on any predicate of the conjunction.)*

By introducing negations and disjunctions, we go beyond Horn Logic, which makes *modus ponens* an incomplete inference system. As we demonstrated in Chapter III, an ontology database with not-gadgets can support limited forms of disjunctive queries by unioning over atomic disjunctions. The unions return more answers than typical databases using the closed-world assumption, because additional inferences can be drawn over explicit negative knowledge via the triggers on negative tables. However ontology databases cannot guarantee completeness in general because of the limitations of Horn Logic. To guarantee completeness, we would need to consider methods including *reductio ad adsurdum* (also known as proof-by-contradiction), case analysis (also known as or-elimination), or resolution as core reasoning faculties – which goes beyond the scope of our investigation.

Another interesting byproduct of not-gadgets is that the exclusion dependency raises an error whenever an inconsistency is detected for the not-gadget. In combination with the forward-reasoning capability of the trigger rules, this proves

to be a very powerful tool for detecting inconsistencies using ontology database. We illustrate this feature using the following case study from biomedical informatics.

Case Study: the Serotonin Example

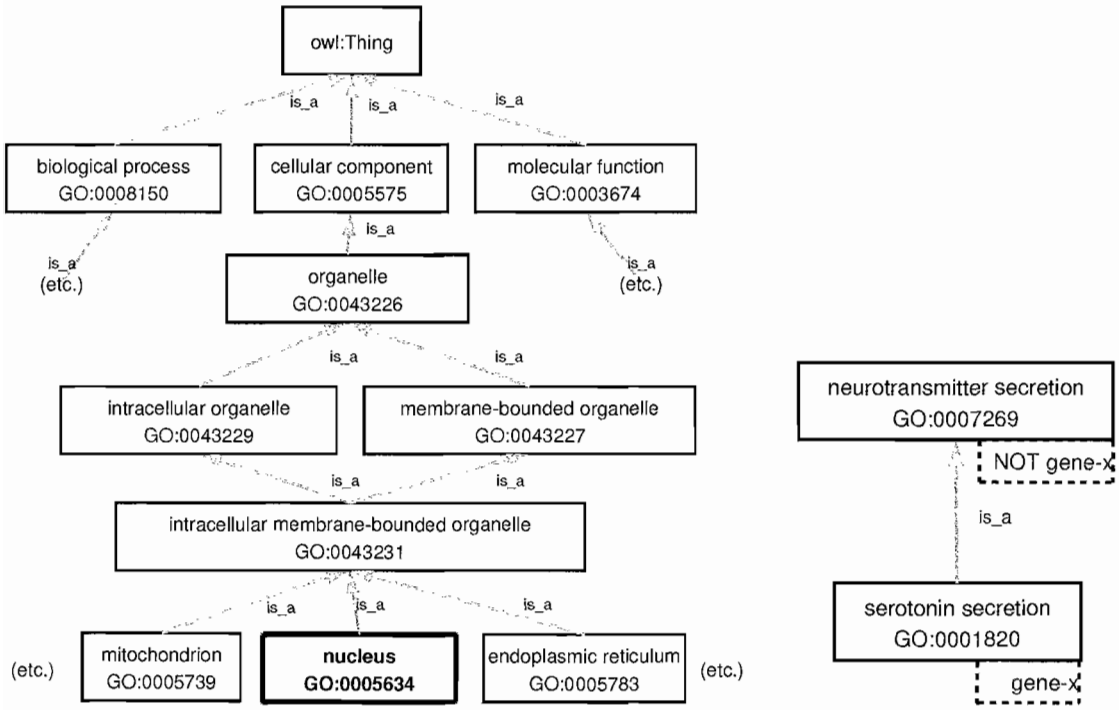
Background

The Gene Ontology (GO) [34, 35] is used specify the molecular functions, biological processes and cellular locations of gene products for the purpose of facilitating information sharing, data fusion and exchange [43] among biological databases including the model organism databases. The Open Biomedical Ontologies (OBO) specification of the GO has on the order of 30,000 concepts arranged in a directed, acyclic graph using mainly two kinds of relationships, "is_a" (sub-class) and "part_of" relationships, forming nearly 40,000 links among concepts[1]. Figure 11(a) shows where the GO term "nucleus" falls in the GO.

Because the GO is relatively simple and the concept hierarchy is mostly limited to an average depth of about 8 and a maximum depth of 14[2], reasoning over the general GO structure is actually not hard at all. Well known transitive closure algorithms suffice and existing Semantic Web reasoners work well enough [42]. The problem is the number of gene annotations is several orders of magnitude beyond

[1]Since first preparing our research data, there are now over 48,496 edges as of April 1, 2009; 44,883 of which are is_a and part_of relationships.

[2]Taken April 1, 2009.



(a) The GO concept graph for nucleus.

(b) The serotonin example.

FIGURE 11: Examples from the Gene Ontology.

what most reasoners can handle, totalling nearly 27 million in March of 2009 when we prepared our research data and growing at a tremendous rate[3].

What makes GO annotations especially interesting to us is that explicit, negative knowledge is also annotated based on experimental results, such as "lzic is *not* involved in beta-catenin binding[4]." But negative data are clearly in the minority. Compared to the 91,000 or so positive assertions from the Zebrafish Information Network (ZFIN) [68] data we used, only 40 were negative facts; only 292 out of 154,000 facts from Mouse Genome Informatics (MGI) [10] data were negative[5].

A negative annotation means a gene does not belong to the specified class within the context of a given experiment. As our results confirm, a strong interpretation of the *not* qualifier leads to contradictions that do not take the biology into account. For example, biologists might observe directly from experimentation that the *p2rx2* gene is not involved in the molecular function *ATP-gated cation channel activity* in the zebrafish[6]. However, in the same experiment, when considered in the context of another gene, *p2rx2* gets a positive annotation for the same GO-ID. One way biologists infer this kind of knowledge is by using mutants which specifically disrupt the function of the specific gene (loss of

[3]Recent reports as of April, 2009 have annotations reaching over 40 million in number.

[4]`http://zfin.org/cgi-bin/webdriver?MIval=aa-markerview.apg&OID=ZDB-GENE-040718-342`

[5]Taken January, 24 2009.

[6]`http://zfin.org/cgi-bin/webdriver?MIval=aa-markergoview.apg&OID=ZDB-GENE-030319-2`

function assay). Another way is to make inferences by adding the specific gene to an accepted assay (gain of function assay).

Therefore, inconsistencies can point to any of the following possible causes: (1) the nature of biology is simply ripe with exceptions, (2) the annotation may be incorrect (e.g., a typographical or curation error), (3) the experimental results were anomalous, or (4) the ontology is incorrect or incomplete. This raises some interesting problems of how to deal with different kinds of inconsistencies, some of which may be admissible, a kind of paraconsistent logic [66].

The Serotonin Example

Hill *et al.* specify a concrete and illustrative example of a recently discovered flaw in the GO, which we refer to as the *serotonin example* [43]. This problem arose particularly because of the interaction between positive and negative annotations and their implications for the consistency of the type hierarchy in general:

> "[GO annotations sometimes] point to errors in the type-type relationships described in the ontology. An example is the recent removal of the type serotonin secretion as an is_a child of neurotransmitter secretion from the GO Biological Process ontology. This modification was made as a result of an annotation from a paper showing that serotonin can be secreted by cells of the immune system where it does not act as a neurotransmitter."

In other words, the GO ontology serves as the most current understanding of the world of genetics as far as the biologists know it to be. As biological knowledge changes, so must the model. Hill *et al.* explain how difficult it is for gene scientists

to detect such data-driven inconsistencies in the GO, leaving it as an open problem to find ways to identify type-type inconsistencies based on annotations from the model organism databases such as ZFIN and MGI.

Figure 11(b) illustrates the inconsistency arising from the serotonin example. In it, some gene (call it "gene-x") was annotated as both being an instance of *serotonin secretion* while *NOT* being an instance of *neurotransmitter secretion*, causing the logical inconsistency based on the type-type (i.e., is_a) hierarchy.

The serotonin example was easily detected by the biologist making the annotation, probably because these concepts are well-known and so closely related, and, furthermore, the annotation spanned a single experimental curation result. However, it is quite possible that inconsistencies due to positive and negative annotations in concepts that are, say, 14 relationships apart would easily go unnoticed by humans – more so if the conflicting annotations span different experiments, publications, curation attempts, or species of model organism. Therefore, gene scientists are motivated to find such logical inconsistencies using automated methods that can scale to large number of instances described by medium- to large-sized ontologies[7].

[7]The GO might be characterized as medium-sized.

Experiment and Results

An unremarkable laptop computer with a 1.8 GHz Centrino processor and 1GB of RAM was used to process the GO ontology in OBO format and generate the corresponding MySQL database schema representing the GO ontology database with negations. We used the OWL-API[8], for this purpose. An unremarkable desktop system with a 1.8 GHz Pentium Processor and 512MB of RAM running Ubuntu Linux was used as the MySQL database server.

We processed annotations from both the Zebrafish Information Network (ZFIN) [68] and the Mouse Genome Informatics (MGI) [10] databases. Only is_a relationships were implemented, as our goal was specifically to detect type-type inconsistencies. As described above, we used a decomposition storage model, therefore every GO-ID became a table, and every is_a relationship between GO-IDs corresponds to a trigger which forward-propagates any insertion on the child class to its parent. Furthermore, every GO-ID has a negative table as well, denoted with an underscore "_" prefix. Between each table and its negative counterpart exists an integrity constraint which ensures mutual exclusion or disjointedness. The exclusion dependency was also implemented as a trigger on both the positive and negative tables, which logs the error to a special table called "_log". One limitation of MySQL is to only allow one trigger per table per event. Therefore, we had to

[8] `http://owlapi.sourceforge.net/`

implement multiple triggers in one in the case of multi-inheritance paths and exclusion dependencies, which was not difficult to implement.

On January 24, 2009, we downloaded the Gene Ontology in OBO format from the GO website[9], as well as annotations from ZFIN and MGI[10]. At the time, there were 28,007 GO-IDs and 38,557 is_a relationships among them. The annotation files have 15 delimited fields, of which we only considered the gene symbol, qualifier (e.g., NOT), and GO-ID. The ZFIN annotations contained 91,147 positive and 42 negative unique annotations of this sort. The MGI annotations contained 130,979 positive and 284 negative unique annotations of this sort. Ignoring the other 12 annotation fields is a gross over-simplification, which we discuss further below.

The steps involved in loading the GO plus annotations to detect inconsistencies are: (1) run the OBO ontology through our tool to create the ontology database with negations schema; (2) load the schema into the MySQL database; (3) pre-process the ZFIN and MGI annotations to create SQL insert statements, where the gene symbol is the value, the GO-ID is the table, and the NOT qualifier indicates using the negative table; (4) load the annotations in to the MySQL database; (5) check the _log table for the type-type inconsistencies detected.

Loading the GO ontology database with negations schema took approximately 32 minutes. We then loaded annotations in this order: (1) ZFIN

[9]Taken on January 22, 2009.

[10]`http://www.geneontology.org/gene-associations/`

positive and ZFIN negative, then (2) MGI positive, finally (3) MGI negative. The results are summarized as follows (specific inconsistencies will be reported and discussed separately in a bioinformatics journal):

1. Loading the ZFIN positive annotations takes 38 minutes. Loading negative ZFIN annotations takes 4 seconds. Nine inconsistencies were logged after loading just the ZFIN positive and negative annotations.

2. Loading the MGI positive annotations takes 42 minutes. Twelve new inconsistencies were logged after loading the MGI positive annotations, meaning there were inter-species inconsistencies with the ZFIN negative annotations.

3. Loading MGI negative annotations takes 20 seconds. Fifty-four new inconsistencies were logged after loading the MGI negative annotations, meaning there were conflicts with both ZFIN and MGI positive annotations.

In total, we found 75 logic inconsistencies among ZFIN and MGI annotations using the GO ontology database with negations technique. We confirmed these results are the complete set of inconsistencies by using the GOOSE[11] database directly. The observed logic inconsistencies fell into three categories. We provide the following examples:

[11] `http://www.berkeleybop.org/goose`

1. Intra-species logic inconsistencies between experimentally supported manual annotations: The zebrafish *p2rx2* gene[12] is annotated as having (inferred from a genetic interaction) and *not* having (inferred from a direct assay) ATP-gated cation channel activity (GO:0004931).

2. Inter-species logic inconsistencies between experimentally supported manual annotations: The zebrafish *bad* gene[13] is annotated (inferred from a direct assay) as *not* being involved in the positive regulation of apoptosis (GO:0043065) in the zebrafish. Meanwhile, annotation of the corresponding mouse gene, *Bad*, indicates it *is* involved in this biological process for the mouse (inferred from a mutant phenotype).

3. Logic inconsistencies between experimentally supported manual annotations and automated electronic annotations (between or within species): The zebrafish *lzic* gene[14] has been electronically annotated (inferred by electronic annotation) as having the function beta catenin binding (GO:0008013) and also *not* having the function beta catenin binding (inferred from physical interaction).

---

[12] http://zfin.org/cgi-bin/webdriver?MIval=aa-markergoview.apg&OID=ZDB-GENE-030319-2

[13] ZFIN:http://zfin.org/cgi-bin/webdriver?MIval=aa-markergoview.apg&OID=ZDB-GENE-000616-1, MGI:http://www.informatics.jax.org/javawi2/servlet/WIFetch?page=markerGO&key=33374

[14] http://zfin.org/cgi-bin/webdriver?MIval=aa-markergoview.apg&OID=ZDB-GENE-040718-342

We discussed results of these findings with ZFIN biologists and came to the following general conclusions:

1. Intra-species logic inconsistencies involving annotations generated by automated electronic annotation pipelines (IEA sources) that conflict with experimentally supported manual annotations (e.g., inferred by direct assay), such as the *lzic* example above, suggest that the automated electronic annotation pipeline makes an assertion that is in direct contradiction to experimentally supported data. This suggests that a review and refinement of the electronic annotation pipeline is needed in this case.

2. Some inter-species inconsistencies highlight possibly interesting biological differences between species that warrant further study. Our example of the *bad* gene is one such example between mouse and zebrafish. In this proof of concept study, we have simply compared genes that use the same gene symbol. However, one could imagine adding sophisticated gene clustering algorithms, that do not rely on shared gene symbols, for determining exactly which genes should be directly compared for logic conflicts.

3. Most intra-species inconsistencies are simply the nature of biology, and can often be explained when the full context of the annotations are considered in more detail, such as the *p2rx2* gene. More complex reasoning would be necessary to resolve whether such cases were of biological interest or not.

4. There were no obvious type-type inconsistencies in the ontology itself. Though this is a negative finding, it is a good one in the sense that much effort is expended by the GO ontology editors to structure the ontology carefully to avoid such cases.

The intra-species inconsistencies arising from direct evidence (IDA, IPI, and other sources) versus automated electronic annotations (IEA sources) were of particular significance and constitute an important biological finding. While conflicting annotations from physical evidence alone are difficult to explain because of the nature of biology, conflicts between manual and automated annotations point directly to possible errors in the automated, electronic annotation transfer rules. We reverse engineered this finding to generate a specialized query against the GO Online SQL Environment (GOOSE) for biology researchers to follow-up on for evaluating IEA transfer rules. This specialized SQL query, which generates precisely the conflicts we discovered has been submitted to the Gene Ontology consortium [15]. As for the inconsistencies that arise because of the nature of biology, this raises some very interesting problems of admissible types of inconsistencies, but we leave this as a future direction in paraconsistent logics [66].

[15] `http://sourceforge.net/tracker/?func=detail&aid=2686444&group_id=36855&atid=469833`

Discussion

In ontology databases with negations and not-gadgets, the semantics of a database *delete* operation became interesting. Deletion can be described using the **K** modal logic operator, just as Reiter did for integrity constraints in [63]. The **K** can be interpreted as meaning "know." A deletion is an assertion that we *do not know* something is true, i.e., $\neg \mathbf{K}C(a)$. Whereas, a negation is an assertion that we know something is not true, i.e., $\mathbf{K}\neg C(a)$. If we treat tuples in a relational database as statements about what is known, such as $\mathbf{K}C(a)$, then the CWA assumption is simply the axiom: $\neg \mathbf{K}C \equiv \mathbf{K}\neg C$. It turns out that Donini *et al.* made some similar observations in [23].

In scientific applications, this distinction between deletion and negation is important since it is often the case that we would like to distinguish between what is assumed to be false (resp., true) and what we *know* to be false (resp., true), as in hypotheses versus empirical evidence. Unlike the open-world assumption which seeks out truth in all possible worlds, ontology databases with negations give us something that remains concrete and constructive.

Theorem 5.0.9. *A trigger-based ontology database has a distinctly different operational semantics from a view-based implementation with respect to deletions.*

Proof. By counterexample, assert the following in the given order: A→B, insert A(a), delete A(a). Now ask the query B(?x). A trigger-based implementation returns "{x/a}." A view-based implementation returns "null." □

Therefore, with respect to ontology databases with negations, we can say definitively that a trigger-based approach is distinctly different from a materialized view-based approach. Indeed, we go as far as to claim an ontology-based approach with not-gadgets is more expressive than views. Consider the following example: assert A→B, insert A(a), negate B(a), now ask the query B(?x). A view-based approach returns "{x/a}" whereas a trigger-based approach raises a contradiction. This example points to an interesting problem in which views entangle assertions, inferences, rule inverses and the contrapositive, but triggers allow for careful differentiation among these logical features.

CHAPTER VI

INFORMATION INTEGRATION

The event-driven architecture of ontology databases using triggers can be extended to integrate two KBs. The key idea is to use namespaces to distinguish terms from each KB and thus safely union concepts between two (or more) ontologies using what are called *bridging axioms* to create a *merged ontology.* Each KB namespace corresponds to the relational database prefix in the corresponding ontology database. If *bridging axioms* in the *merged ontology* are Horn-like rules, then we can implement them using triggers as we normally would for any other axiom, adding the database prefix corresponding to each namespace.

In this chapter, we adopt the theory of *inferential information integration* which includes the definition of *merged ontology* and *bridging axiom* as well as the OntoEngine and OntoGrate reasoning systems developed by Dou *et al.* [25, 26, 27, 28] and extend that framework to include data translation across ontology databases. Our main contributions are to define *inferential ontology*

database exchange, and to illustrate and discuss how to implement this process by using a manufactured example.

Inferential Ontology Database Exchange

In inferential information integration, query translation and data translation are formally defined as logical entailments with respect to a merged ontology having bridging axioms. By performing sound inference over the merged ontology's bridging axioms, the entailments under a target ontology can be inferred automatically (because $KB \vdash \phi$ implies $KB \models \phi$). This method works well for our purposes because we only require sound (not complete) inference to achieve our desired results.

A merged ontology is similar to the notion of a global view over local schemas (global-as-view) [50] for data integration. It consists of the union of elements from a source and target ontology but also defines the semantic mappings between them as bridging axioms. A merged ontology allows all the relevant symbols in a domain to interact so that facts can be translated from one ontology to another using inference over the bridging axioms. Discovering bridging axioms is difficult and some claim that it cannot be fully automated. However, there are some semi-automatic systems including ours [59].

Query translation is one way to integrate data. It applies mostly to scenarios in which one system mediates queries among various sources. A typical example

would be a federated database [65]. On the other hand, *data translation* is more common to migration or exchange scenarios, where information is exported from one location to another [48]. A typical example would be a data warehouse [6]. We extend this idea to ontology databases by defining the following:

Definition 6.0.10 (Inferential Ontology Database Exchange). *Let $\mathcal{S}$ be an ontology database source and $\mathcal{T}$ be an ontology database target specified by ontologies O_S and O_T, respectively. Let $\mathcal{M} = (O_S, O_T, \Sigma)$ be a merged ontology containing bridging axioms Σ, such that Σ is a set of Horn rules relating terms from O_S and O_T qualified with namespaces. Let d_S be a set of tuples in $\mathcal{S}$. The inferential ontology database exchange of d_S is the largest set of assertions d_T entailed by d_S with respect to $\mathcal{M}$.*

By simply extending ontology databases with the corresponding namespace prefixes (supported by most DBMS), we can translate and exchange any data asserted under a source ontology database $\mathcal{S}$ into data under $\mathcal{T}$ using ECA triggers as usual for each bridging axiom expressed in Horn form. As data is inserted into a table in $\mathcal{S}$, the relevant event is detected, fires the appropriate triggers, and inserts the corresponding data into $\mathcal{T}$. Because our methodology implements the typical forward-chaining algorithms for Horn Logic, our translation constitutes the sound inference required.

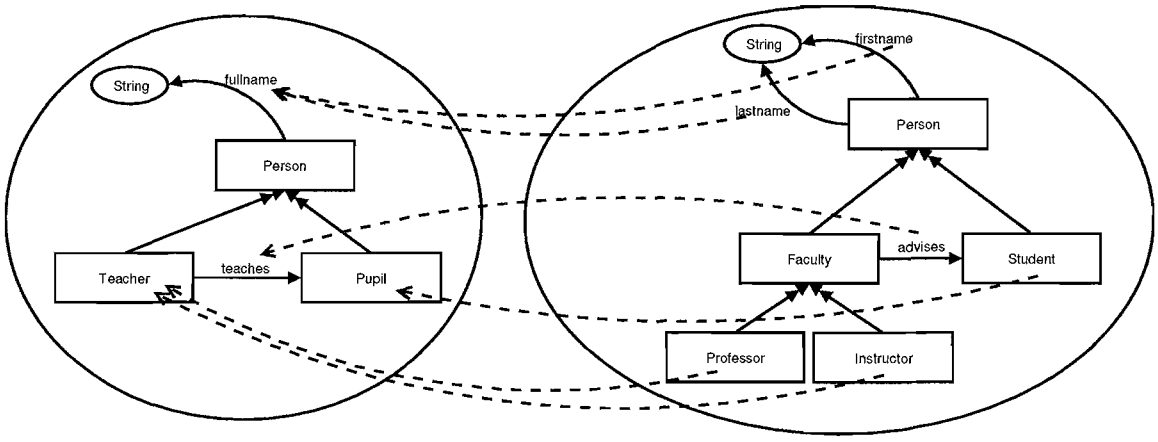


FIGURE 12: Merged Teacher-Student ontologies.

Case Study: Data Exchange

We implemented two ontologies in the Teacher-Student domain and defined a merged ontology using bridging axioms as depicted in Figure 12. In the figure, the ontology on the right which uses the Faculty term is considered the source ontology (based on the direction of the mappings we defined), whereas the other ontology using the Teacher term is the target ontology. For example, there is an axiom relating first and last name in the source ontology to full name in the target ontology. That axiom can be expressed as a rule using the built-in string concatenation function:

$$\forall x, y, z.\ \text{S:}firstname(x, y) \wedge \text{S:}lastname(x, z)$$

$$\Rightarrow \text{T:}fullname(x, \text{concat}(y, \text{`` '', } z))$$

Because the rule is a conjunction, we implement it as a set of triggers, one for each conjunct. The reason we need two triggers is that satisfying either predicate in the conjunct could potentially fire the rule, so we need to set a listener on each predicate (i.e., table) which checks to see if the other predicate is also satisfied. The triggers are written in MySQL syntax loaded as part of the source ontology database as follows:

```
CREATE TRIGGER trg_fn_AFTER_INSERT AFTER INSERT ON firstname
FOR EACH ROW
trig:BEGIN
 -- enforce bridging axiom: (mysource)fn + ln -> (mytarget)n
 INSERT INTO mytarget.fullname(subject,object)
   SELECT fn.subject, concat(fn.object, ' ', ln.object)
   FROM mysource.firstname fn, mysource.lastname ln
   WHERE fn.subject = ln.subject
   AND fn.subject = NEW.subject;
END trig

CREATE TRIGGER trg_ln_AFTER_INSERT AFTER INSERT ON ln
FOR EACH ROW
trig:BEGIN
 -- enforce bridging axiom: (mysource)fn + ln -> (mytarget)n
```

```
    INSERT INTO mytarget.fullname(subject,object)
      SELECT fn.subject, concat(fn.object, ' ', ln.object)
      FROM mysource.firstname fn, mysource.lastname ln
      WHERE fn.subject = ln.subject
      AND ln.subject = NEW.subject;
  END trig
```

Similar to what we did when testing load-time performance, we wrote a program which generated some uniformly distributed extensional data instances for the source ontology. Both ontologies are small enough to have a negligible schema load time, taking under 100 milliseconds on average to load their respective ontology database schemas. We generated four sets of data instances under the source ontology semantics, each dataset greater than the last by a factor of ten (i.e., we used a logarithmic scale). We measured the total time it takes to load the dataset into the source ontology database. Because the source ontology database contains the regular ontology rules together with the bridging axioms rules, the total time measured includes three major parts: (1) the time it takes to forward propagate data within the source ontology database, (2) the time it takes to forward propagate data across to the target ontology database (via the bridging axioms), and (3) the time it takes to forward propagate data within the target ontology database. Figure 13 summaries the performance results, which shows the near-linear

performance we expect from small ontologies (based on our prior load-time observations).

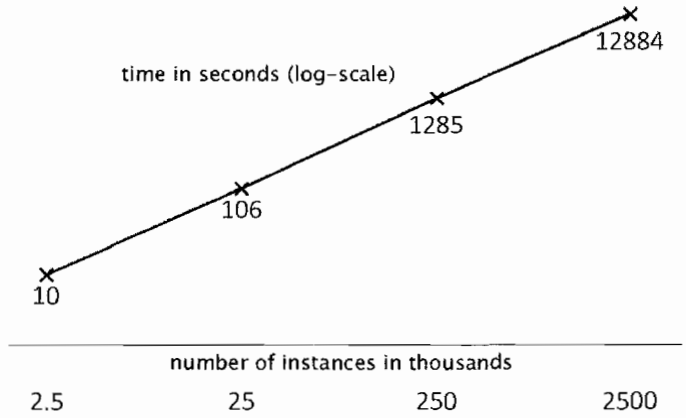


FIGURE 13: Integration Performance for Ontology Databases.

Discussion

Our research was inspired by prior inferential data integration work. By lifting a database schema into an ontology, we reduced the problem of database integration to ontology translation. The question that arose out of this work was whether we can reverse the process: *Can we "ground" an ontology into a database schema?* Originally, that question was simply focused around conceptual modeling using ontologies. However, this case study shows that the very question has direct ramifications to integration as well. That is, where the lifting process significantly improves upon query answering performance, the grounding process significantly improves on data translation scalability.

Using the same merged ontology, the OntoEngine inference engine shows significant trouble managing large amounts of data, which is consistent with observations by Guo *et al.* [39, 40] on the trouble that memory-based KB systems experience. Essentially, they run out of memory. Figure 14 illustrates how OntoEngine performs over the same merged ontologies and datasets used above. In the figure, we indicated that we needed to increase available memory to 4 GB of RAM in order for the reasoner to process the 250k dataset. Even after increasing to the maximum 8 GB of RAM on the system, the reasoner crashed for the 2.5M dataset.

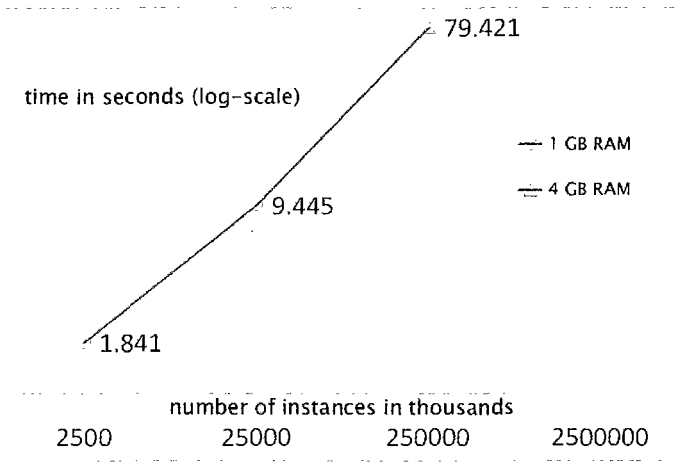


FIGURE 14: Integration Performance for OntoEngine.

The conclusion that we would like to highlight from this study goes back to ideas discussed earlier in Chapter IV: that persistent KBs that use underlying, efficient database optimization features can scale far beyond traditional memory-based KB systems. If we recall from Chapter II, one of the major

assumptions that Reiter made in the context of deductive query answering is that space is limited and should be balanced against computational power. Our work challenges this assumption: disk-space is unlimited and should be leveraged against limited computational power. Moreover, our implementation is extremely simple compared to the programming that would be required to implement the sophisticated caching mechanisms already present in database management systems. Finally, our implementation plays to the lowest common denominator for the majority of relational database management engines. That is, we do not require sophisticated materialized and updateable view features. Our work combines and builds on the strengths of existing technologies in a simple and elegant fashion.

CHAPTER VII

CONCLUSION

We presented a mapping from ontologies to databases such that a KB system based on a given ontology can be embedded inside a relational database management system. The mapping makes instance checking using ontology databases sound and complete for ontologies based on Horn Logic. When not-gadgets are included, we can also perform inconsistency detection. This methodology has been implemented as a tool which can automatically re-purpose off-the-shelf relational database management systems, such as MySQL, for reasoning over Semantic Web knowledge bases.

The main problem we aimed to address by developing this methodology is the poor scalability of reasoning systems over very large numbers of instances. Our work differs from that of others because we take advantage of more features of relational databases such as views, triggers and integrity constraints to implement not only subsumption-like reasoning for instance checking, but also satisfiability-like reasoning for inconsistency detection. Furthermore, unlike other similar approaches

aimed at addressing the scalability of RDF stores, we do not use an external theorem prover to perform any reasoning. Our main contribution hinges on the updated assumption that space is more expendable than it has been in the past.

In addition to scalability, we also aimed to increase the expressiveness of existing database-oriented techniques by extending Horn-like logics with negations. Although we lose completeness, we are still able to solve an important open problem in biomedical informatics. We have argued that there is a tradeoff not only between expressiveness and tractability, but that the tradeoff is three-fold and includes completeness. We hope that going forward the Semantic Web community will consider relaxing the completeness requirement in order to solve more interesting and relevant reasoning problems.

We applied our technique toward several case studies to better understand and demonstrate the capabilities of our tool. Firstly, we compared the scalability of our system against a well-studied and similar approach, the DLDB system, and followed up with new studies and additional benchmarks on the surprisingly low load-time costs of our methodology. Secondly, we demonstrated the promise that ontology databases holds for our driving biomedical project, NEMO. Thirdly, we illustrated other possible uses for ontology databases, such as for inconsistency detection, in the serotonin example. Finally, we demonstrated that our tool can also be used to perform highly scalable information integration between two KB systems.

Future Work

Our next steps will be to extend the event-driven framework used here for information integration to enable distributed information integration using ontology databases. That is, we expect that KBs can reside locally on systems that are distributed across a network. Bridging axioms and namespaces can be used to relate terms across the ontologies. As data is asserted on a local ontology database, the bridging axioms will be fired in an event-driven manner across the network to the remote ontology database.

To realize this vision, we require two main extensions to our work so far. Firstly, we need to develop a message-passing protocol that registers the bridging axioms that will negotiates data exchange between the source and target KBs. The protocol can be based on the publish-subscribe paradigm [31] which fits well with the asynchronous nature of inference rule application. That is, the ordering of rule applications (just as the ordering of events) is irrelevant. Secondly, we need a mechanism for buffering the incoming data from remote KBs so that they do not necessarily contend with reasoning over the local data store. Again because rule application is asynchronous, we can employ the persistent queue methodology outlined in [14] for this purpose. Queues are often used in forward chaining algorithms. In fact, the persistent queue can be implemented as a buffer for reasoning over the local repository as well, especially in scenarios where the subsumption depth is extremely large (causing the slowdown we demonstrated in

Chapter IV) and we do not want reasoning to interfere with other database tasks. In other words, reasoning can become a background task which operates over a persistent queue as resources are available. What would be an interesting research question is whether, by using a background process together with persistent queues, we can extend the expressiveness of the ontology database methodology even further to include cyclic (yet terminating) terminologies.

BIBLIOGRAPHY

[1] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. SW-Store: A Vertically Partitioned DBMS for Semantic Web Data Management. *VLDB Journal*, 18(2):385–406, April 2009.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[3] F. Baader, C. Lutz, and B. Suntisrivaraporn. Is Tractable Reasoning in Extensions of the Description Logic $\mathcal{EL}$ Useful in Practice? In *Methods for Modalities Workshop*, 2005.

[4] F. Baader and W. Nutt. Basic Description Logics. In *Description Logic Handbook*, pages 43–95, 2003.

[5] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[6] P. A. Bernstein and E. Rahm. Data warehouse scenarios for model management. In *ER*, pages 1–15, 2000.

[7] O. Bodenreider, B. Smith, A. Kumar, and A. Burgun. Investigating subsumption in snomed ct: An exploration into large description logic-based biomedical terminologies. *Artificial Intelligence in Medicine*, 39(3):183–195, 2007.

[8] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *International Semantic Web Conference (ISWC'02)*, pages 54–68, 2002.

[9] A. P. Buchmann, H. Branding, T. Kudrass, and J. Zimmermann. Reach: a real-time, active and heterogeneous mediator system. *IEEE Data Eng. Bull.*, 15(1-4):44–47, 1992.

[10] C. J. Bult, J. T. Eppig, J. A. Kadin, J. E. Richardson, and J. A. Blake. The Mouse Genome Database (MGD): mouse biology and model systems. *Nucleic acids research*, 36(Database issue), January 2008.

[11] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable Description Logics for Ontologies. In *National Conference on Artificial Intelligence (AAAI'05)*, pages 602–607, 2005.

[12] S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. Constraint enforcement through production rules: Putting active databases at work. *IEEE Data Eng. Bull.*, 15(1-4):10–14, 1992.

[13] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *International Conference on Very Large Data Bases (VLDB'91)*, pages 577–589, 1991.

[14] S. Ceri and J. Widom. Managing semantic heterogeneity with production rules and persistent queues. In *VLDB*, pages 108–119, 1993.

[15] S. Chakravarthy, E. N. Hanson, and S. Y. W. Su. Active data/knowledge bases research at the university of florida. *IEEE Data Eng. Bull.*, 15(1-4):35–39, 1992.

[16] D. D. Chamberlin, M. M. Astrahan, M. W. Blasgen, J. Gray, W. F. King III, B. G. Lindsay, R. A. Lorie, J. W. Mehl, T. G. Price, G. R. Putzolu, P. G. Selinger, M. Schkolnick, D. R. Slutz, I. L. Traiger, B. W. Wade, and R. A. Yost. A history and evaluation of system r. *Communications of the ACM*, 24(10):632–646, 1981.

[17] P. P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *Transactions on Database Systems*, 1(1):9–36, 1976.

[18] V. Christophides, G. Karvounarakis, D. Plexousakis, M. Scholl, and S. Tourtounis. Optimizing taxonomic semantic web queries using labeling schemes. In *Journal of Web Sematics*, volume 1, pages 207–228. Elsevier, 2004.

[19] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[20] G. P. Copeland and S. N. Khoshafian. A decomposition storage model. In *International Conference on Management of Data (SIGMOD'85)*, pages 268–279, 1985.

[21] O. Curé and R. Squelbut. A Database Trigger Strategy to Maintain Knowledge Bases Developed Via Data Migration. In *Portuguese Conference on Artificial Intelligence*, pages 206–217, 2005.

[22] S. W. Dietrich, S. D. Urban, J. V. Harrison, and A. P. Karadimce. A dood ranch at asu: Integrating active, deductive and object-oriented databases. *IEEE Data Eng. Bull.*, 15(1-4):40–43, 1992.

[23] F. M. Donini, D. Nardi, and R. Rosati. Description logics of minimal knowledge and negation as failure. *Transactions on Computational Logic*, 3(2):177–225, 2002.

[24] D. Dou, G. Frishkoff, J. Rong, R. Frank, A. Malony, and D. Tucker. Development of NeuroElectroMagnetic Ontologies (NEMO): A Framework for Mining Brainwave Ontologies. In *International Conference on Knowledge Discovery and Data Mining (KDD'07)*, pages 270–279, 2007.

[25] D. Dou and P. LePendu. Ontology-based Integration for Relational Databases. In *Symposium on Applied Computing*, pages 461–466, 2006.

[26] D. Dou, P. LePendu, S. Kim, and P. Qi. Integrating Databases into the Semantic Web through an Ontology-based Framework. In *International Workshop on Semantic Web and Databases*, page 54, 2006.

[27] D. Dou, D. V. McDermott, and P. Qi. Ontology Translation on the Semantic Web. *Journal of Data Semantics*, 2:35–57, 2005.

[28] D. Dou, J. Z. Pan, H. Qin, and P. LePendu. Towards Populating and Querying the Semantic Web. In *International Workshop on Scalable Semantic Web Knowledge Base Systems*, pages 129–142, 2006.

[29] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems, 2nd Edition.* Benjamin/Cummings, 1994.

[30] O. Etzion. PARDES: a Data-Driven Oriented Active Database Model. *ACM SIGMOD Record*, 22(1):7–14, 1993.

[31] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[32] G. Frishkoff, P. LePendu, R. Frank, H. Liu, and D. Dou. Development of Neural Electromagnetic Ontologies (NEMO): Ontology-based Tools for Representation and Integration of Event-related Brain Potentials. In *International Conference on Biomedical Ontology*, pages 31–34, 2009.

[33] G. A. Frishkoff. Hemispheric differences in strong versus weak semantic priming: Evidence from event-related brain potentials. *Brain Lang*, 100(1), 2007.

[34] Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[35] Gene Ontology Consortium. The Gene Ontology (GO) project in 2008. *Nucleic Acids Research*, 36(Database issue), January 2008.

[36] C. Goble and R. Stevens. State of the Nation in Data Integration for Bioinformatics. *Journal of Biomedical Informatics*, February 2008.

[37] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[38] N. Guarino. Formal Ontology in Information Systems. In *International Conference on Formal Ontology in Information Systems.*

[39] Y. Guo, Z. Pan, and J. Heflin. An Evaluation of Knowledge Base Systems for Large OWL Datasets. Technical report lu-cse-04-012, Lehigh University, CSE Department, 2004.

[40] Y. Guo, Z. Pan, and J. Heflin. An Evaluation of Knowledge Base Systems for Large OWL Datasets. In *International Semantic Web Conference (ISWC'04)*, pages 274–288, 2004.

[41] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005.

[42] V. Haarslev and R. Möller. High Performance Reasoning with Very Large Knowledge Bases: A Practical Case Study. In *International Joint Conferences on Artificial Intelligence (IJCAI'01)*, pages 161–168, 2001.

[43] D. P. Hill, B. Smith, M. S. McAndrews-Hill, and J. A. Blake. Gene Ontology annotations: what they mean and where they come from. *BMC Bioinformatics*, 9(5), 2008.

[44] I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The Instance Store: DL Reasoning with Large Numbers of Individuals. In *Description Logics*, 2004.

[45] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: the making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[46] R. Hull and R. King. Semantic database modeling: survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[47] T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

[48] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Symposium on Principles of Database Systems (PODS'05)*, pages 61–75, 2005.

[49] J. Lee and R. Goodwin. Ontology Management for Large-Scale E-Commerce Applications. In *International Workshop on Data Engineering Issues in E-Commerce*, pages 7–15, 2005.

[50] M. Lenzerini. Data Integration: a Theoretical Perspective. In *Symposium on the Principles of Database Systems (PODS'02)*, pages 233–246, 2002.

[51] P. LePendu, D. Dou, G. A. Frishkoff, and J. Rong. Ontology Database: a New Method for Semantic Modeling and an Application to Brainwave Data. In *International Conference on Statistical and Scientific Database Management*, pages 313–330, 2008.

[52] P. LePendu, D. Dou, and D. Howe. Detecting Inconsistencies in the Gene Ontology using Ontology Databases with Not-gadgets. In *International Conference on Ontologies, Databases and Application of Semantics*, pages 948–965, 2009.

[53] H. J. Levesque and G. Lakemeyer. *The Logic of Knowledge Bases.* MIT Press, 2001.

[54] J. Minker. Logic and databases: A 20 year retrospective. In *International Workshop on Logic in Databases*, pages 3–57, 1996.

[55] B. Motik, I. Horrocks, and U. Sattler. Bridging the Gap Between OWL and Relational Databases. In *International Conference on World Wide Web (WWW'07)*, pages 807–816, 2007.

[56] T. Neumann and G. Weikum. Scalable Join Processing on Very Large RDF Graphs. In *International Conference on Management of Data (SIGMOD'09)*, 2009.

[57] Z. Pan and J. Heflin. DLDB: Extending Relational Databases to Support Semantic Web Queries. In *Workshop on Practical and Scalable Semantic Systems*, 2003.

[58] N. W. Paton and O. Díaz. Active Database Systems. *ACM Computing Surveys*, 31(1):63–103, 1999.

[59] H. Qin, D. Dou, and P. LePendu. Discovering Executable Semantic Mappings Between Ontologies. In *Proceedings of the International Conference on Ontologies, Databases and Application of Semantics*, pages 832–849, 2007.

[60] R. Reiter. Deductive Question-Answering on Relational Data Bases. In *Logic and Data Bases*, pages 149–177, 1977.

[61] R. Reiter. On Closed World Data Bases. In *Logic and Data Bases*, pages 55–76, 1977.

[62] R. Reiter. Towards a Logical Reconstruction of Relational Database Theory. In *On Conceptual Modelling—Perspectives from Artificial Intelligence, Databases, and Programming Languages*, pages 191–233. 1984.

[63] R. Reiter. What should a database know? In *Symposium on Principles of Database Systems (PODS'88)*, pages 302–304, 1988.

[64] R. Reiter. What Should a Database Know? *Journal of Logic Programming*, 14(1&2):127–153, 1992.

[65] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[66] J. K. Slaney. Relevant Logic and Paraconsistency. In *Inconsistency Tolerance*, pages 270–293, 2005.

[67] B. Smith and C. Welty. Ontology–Towards a New Synthesis. In *International Conference on Formal Ontology in Information Systems*. ACM, 2001.

[68] J. Sprague, L. Bayraktaroglu, Y. Bradford, T. Conlin, N. Dunn, D. Fashena, K. Frazer, M. Haendel, D. G. Howe, J. Knight, P. Mani, S. A. Moxon, C. Pich, S. Ramachandran, K. Schaper, E. Segerdell, X. Shao, A. Singer, P. Song, B. Sprunger, C. E. Van Slyke, and M. Westerfield. The Zebrafish Information Network: the zebrafish model organism database provides expanded support for genotypes and phenotypes. *Nucl. Acids Res.*, 36:D768–72, November 2007.

[69] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988.

[70] O. Vasilecas and D. Bugaite. An algorithm for the automatic transformation of ontology axioms into a rule model. In *International Conference on Computer Systems and Technologies*, pages 1–6, 2007.

[71] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based Integration of Information – a Survey of Existing Approaches. In *Workshop on Ontologies and Information Sharing (IJCAI'01)*, pages 108–117, 2001.