Detecting Inconsistencies in the Gene Ontology Using Ontology Databases with Not-gadgets

Paea LePendu[1], Dejing Dou[1], and Doug Howe[2]

[1] Computer and Information Science, University of Oregon, Eugene OR 97403, USA
[2] Zebrafish Information Network, University of Oregon, Eugene OR 97403, USA

Abstract. We present ontology databases with not-gadgets, a method for detecting inconsistencies in an ontology with large numbers of annotated instances by using triggers and exclusion dependencies in a unique way. What makes this work relevant is the use of the database itself, rather than an external reasoner, to detect logical inconsistencies given large numbers of annotated instances. What distinguishes this work is the use of event-driven triggers together with the introduction of explicit negations. We applied this approach toward the serotonin example, an open problem in biomedical informatics which aims to use annotations to help identify inconsistencies in the Gene Ontology. We discovered 75 inconsistencies that have important implications in biology, which include: (1) methods for refining transfer rules used for inferring electronic annotations, and (2) highlighting possible biological differences across species worth investigating.

1 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. Especially evidenced by the explosive growth of annotated scientific biological data, ontologies promise facilitated information sharing, data fusion and exchange among many, distributed and possibly heterogeneous data sources [18].

Typically, reasoning is divided 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 [17], 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, Pan and Heflin [15] has shown that many memory-based and disk-based systems demonstrate significant signs of trouble around 1 million instances and completely fail at around 3 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.

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part II, LNCS 5871, pp. 948–965, 2009.
© Springer-Verlag Berlin Heidelberg 2009

In this paper we present *ontology databases with not-gadgets*, a new approach which distinguishes itself by using several first-order logic features of database management systems in order to directly support the reasoning process, rather than by incorporating an external theorem prover. In addition to capitalizing on integrity constraint checking, our work uniquely explores the use of event-driven, database triggers to perform *modus ponens* over extended Horn-logics in a way that has not been done before, to our knowledge, by introducing explicit negations and not-gadgets into the ontology database structure. Our prior work [21] focused on the implementation, scalability and performance aspects of a trigger-based approach for subsumption reasoning. In contrast, the present work expounds the theoretical background and justification for this approach, extending our system to include negations and therein formally differentiating this approach from typical view-based approaches.

One of the fundamental advantages of using triggers, as in a materialized view, is we can employ the amortization principle: pre-computing the inference upfront to achieve faster query performance, especially when queries occur much more frequently than updates. In addition, triggers appear to be more expressive than views as we note in Section 3.1. Furthermore, the event-driven model, which is a well-known parallel programming design pattern, holds some promise for distributed reasoning systems.

Ultimately, our goal is very much in the style of deduction modulo theories [11] – to separate as much computation from reasoning as possible, leaving to automated theorem provers only the most difficult tasks, such as case analysis over domain closures (i.e., completeness or total participation constraints) which databases cannot seem to support. The work by Motik, Horrocks and Sattler [22] has made a similar case for offloading aspects of reasoning to the database, namely with regard to integrity constraint checking.

In other related work, many advances have been made toward addressing the scalability of reasoning over large numbers of instances. They fall into two main categories. Firstly, some attempt to optimize the characteristically inefficient self-join queries over RDF triple stores, which utilize the vertical storage model such as the famous Sesame system [4,7,23]. Secondly, others attempt to sidestep the costly self-joins by partitioning the vertical model along properties or concepts [2,8,19]. However, in almost every attempt so far, to the best of our knowledge, the database is almost purely used for efficient storage and retrieval, when it is well-known that databases employ sophisticated, logic-based features for integrity maintenance and consistency control.

As a motivation for our work, we have studied a particularly interesting open problem in the biomedical domain, that of detecting inconsistencies in the Gene Ontology (GO) using annotations from the various model organism databases [18], exemplified by the *serotonin example* which we summarize later in this paper. What makes this problem interesting is the understanding that the ontology itself, being defined by humans, may be incorrect, and that the annotations themselves, being based on empirical evidence, may point to such inconsistencies (incorrectness) in the model. Furthermore, this example is

interesting because of the explicit, negative instances that are also empirically tested and annotated by biologists, such as "lzic is *not* involved in beta-catenin binding." We summarize our case study in this paper as well.

The rest of this paper is organized as follows. Section 2 presents the general problem of detecting inconsistencies, discusses related work, covers basic background, and synthesizes the key observations and challenges in addressing this problem. Section 3 outlines our specific solution and main contribution, which is the use of event-driven triggers and not-gadgets in ontology databases. We also draw from this work some insights on the semantics of deletion in Section 3.1 and discuss some general observations and drawbacks of our approach in 3.2. Section 4 presents our case study, which uses gene annotations in order to attempt to find inconsistencies in the GO. We offer concluding remarks and future directions in Section 5.

2 The Problem

The problem is to detect inconsistencies given an ontology with a large number of annotated instances. There are three main challenges to address, which we outline in Section 2.3. Before we highlight those challenges, we first provide some related work in Section 2.1 then summarize additional background in Section 2.2.

2.1 Related Work

Deductive Databases. Reiter's seminal works on logic and databases laid firm groundwork for much of the subsequent research in ontologies, databases and knowledge bases today, including Datalog. He coined the closed-world assumption (CWA), he envisioned databases and inference engines working in concert [24], he reformulated relational database theory in terms of first order logic [25], and he pointed out that the semantics of integrity constraints requires a modal operator [26]. One of the major assumptions Reiter made is that space is limited and he therefore balanced computation against space, but this assumption is less true today.

Ontologies and Databases. DL-Lite [6] makes steps toward bridging the gap between ontologies and databases by defining a complete yet tractable fragment of description logic which closely captures database schema semantics. As previously mentioned, Motik *et al.* [22] offload integrity constraint checking to the database, further bridging the gap. Finally, other recent works have explored the use of database triggers in the context of ontologies for truth maintenance [9], for transforming axioms into rules [31], and for managing large-scale applications [20].

Scalable Knowledge Bases. Connecting ontologies with databases shares many of the same goals as scalable knowledge base systems. The Lehigh University Benchmark [16] provides the necessary framework for evaluating and comparing

systems. The DLDB [15] implementation uses the traditional view-based approach. Other approaches include, as mentioned, optimizations on vertical RDF stores [4,7,23] or partitioned data models [2,8,19].

Information Integration. An important motivation for using ontologies is the promise for integrating information. 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 [14,32].

Ontologies and Bioinformatics. The Gene Ontology Consortium [13] brings together many tools and researchers from several communities including the online model organism databases. Each model organism database, such as for the mouse (MGI) or zebrafish (ZFIN) [5,29], has its own web-based search capability for the species. The National Center for Biomedical Ontologies (NCBO) [27] maintains several important resources, including the Open Biomedical Ontology (OBO) format that is popularly used in bioinformatics as well as the BioPortal which maintains a registry of existing biomedical ontologies and related tools.

2.2 Background

Logic and Databases. We first provide some basic background for the reader on the theory of logical systems. First, please read "$\Gamma \vdash \phi$" as "Γ *derives* ϕ," and read "$\Gamma \models \phi$" as "Γ *entails* ϕ." In a logical system, Γ is a set of formulae together with a proof system for deriving new formulae, such as 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.

These formulations are relevant to database systems because we can think of a database as a logical system. The basic idea is that the internal mechanisms for answering queries constitute a database's proof system, relations are predicates, tuples are ground clauses, and relationships or constraints are axioms. Please see Reiter [25] for the full logical reconstruction of relational database theory.

We can therefore consider the tuples of a database, DB, as derivable from it: DB $\vdash R_i(c_1,c_2...,c_k)$, where R_i is some k-ary relation and $c_1 \ldots c_k$ are constants. Furthermore, the DB derives an answer to a conjunctive query $q(\boldsymbol{x})$ by finding a vector of constants $\boldsymbol{c}$ that substitute for $\boldsymbol{x}$, which we write as $\theta(q(\boldsymbol{x}/\boldsymbol{c}))$. θ is called the substitution set, which we can think of as the answer table to the query q. A conjunctive query is the usual expression of the form $\{\boldsymbol{x} \mid \text{CONJ}(\boldsymbol{x}, \boldsymbol{y})\}$, where the vector of variables, $\boldsymbol{x}$, are distinguished and meant to be bound and returned as answers and $\boldsymbol{y}$ are non-distinguished variables which are existentially qualified and for which the bindings can be discarded once the answer is computed. A DB therefore returns a set of such substitutions θ as answers to a conjunctive query.

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 only permits formulae with at most one positive literal when presented in conjunctive normal form: $\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n \vee q$. We can equivalently view Horn formulae as rules with conjunctions on the left and a single formula on the right: $p_1 \wedge p_2 \wedge \ldots \wedge p_n \rightarrow q$. The main idea of *modus ponens* is to satisfy the left-hand side of a rule in order to conclude the right-hand side:

$$\frac{p \rightarrow q \quad p}{q}$$

Resolution operates without restriction on the number of positive literals and relies on proof-by-contradiction ($p \wedge \neg p$ being unacceptable) to reach a derivation:

$$\frac{p \vee q \quad \neg p}{q}$$

Detecting Inconsistency. Proof-by-contradiction is the key to detecting inconsistency in general, making resolution a natural proof procedure. The main idea is if you apply resolution to the set of all formulae and can only reach a contradiction, then we can conclude the formulae are inconsistent. The problem is resolution introduces non-determinism – which should be the resolvent to apply first, $\neg q$ or $\neg p$? As a result, reasoning is often described as generally exponential. Furthermore, proof-by-contradiction is an non-constructive proof theory, which does not fit naturally with the constructive nature of database systems.

Horn Logic with *modus ponens* fits nicely with database systems because it is a constructive proof theory. We refer the reader to the literature on Datalog for more background [30]. However, by itself, we cannot attack the problem of detecting inconsistency. We therefore extend the expressiveness of Horn Logic slightly to permit negated literals on the left- and right-hand side of rules.

Horn Logic with negations gives rise to inconsistencies whenever, by *modus ponens*, we derive both p and $\neg p$. The problem of detecting inconsistencies therefore reduces to a search (typically called *backward chaining* because we recurse backward along the rules) for a proof of both p and $\neg p$.

Logical Database Features: Views and Triggers. Putting aside negations for a moment, we would like to briefly illustrate how to implement Horn Logic inside of a database. The idea is not new, see Datalog views [30]. First, we structure the schema of the database to reflect the logic as closely as possible, so that a logic query translates directly to a database query. That is, k-ary predicates become k-ary relations of the same name and ground-terms are simply tuples in the corresponding database relation. This is called a decomposition storage model, which has been shown to be at least as effective as other partitioning schemes while retaining many desirable properties, the least of which is straight forward query rewriting [2,8].

For example, the general rule, "All sisters are siblings," together with the fact, "Mary and Jane are sisters," clearly implies that, "Mary and Jane are siblings," by a simple application of the *modus ponens* rule. This form of reasoning, sometimes referred to as instance checking via subsumption in Description Logic [3], can be implemented as a union of views:

```
CREATE VIEW siblings(x,y) as
SELECT x,y FROM a_siblings
UNION
SELECT x,y FROM sisters
```

Where the asserted data, denoted by the "a_" prefix, is distinguished from the inferred data. Every inferred set of data necessarily includes its asserted data (e.g., siblings contains a_siblings and sisters contains a_sisters). We demonstrated in [21] that database triggers can implement the same thing:

```
CREATE TRIGGER subproperty_sisters_siblings
BEFORE INSERT (x,y) INTO sisters
EXECUTE INSERT (x,y) INTO siblings
```

In SQL the query, "Who are the siblings of Jane?," would be:

```
SELECT x FROM siblings WHERE y='Jane'
```

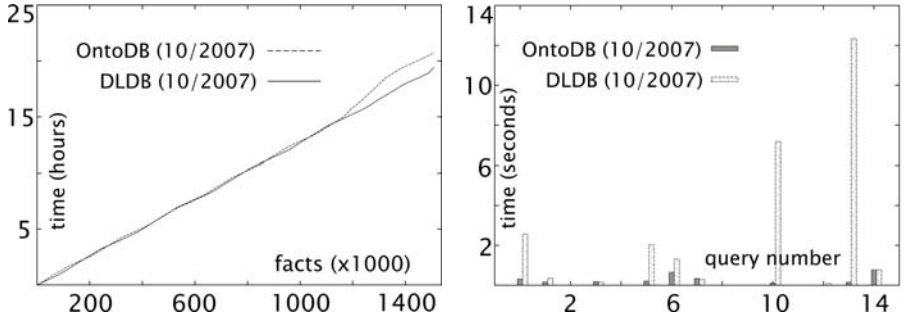
Using either the view-based or trigger-based approach, the answer returned is "Mary" as expected. In [21], we called this family of databases *ontology databases* because they model ontologies and answer ontology-based queries.

We emphasize again that there is no query translation or plan generation necessary because of the decomposition storage model. This makes the system simple, yet highly generalizable [8] without paying a significant price in performance [2].

Key Observations. The fundamental difference between 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 [21] we confirmed by using the LUBM benchmark [16] that by using triggers in this way, performance clearly benefits by several orders of magnitude, but we were surprised to find that it came at very little cost. Load-time, the time it takes to load the asserted data, appeared unaffected. This result surprised us because we expected that forward-propagation would significantly increase load-time. Furthermore, on average, actual disk-space usage merely tripled despite having stored the full transitive closure of all instances.

At a glance, the overlapping trends in Figure 1(a) illustrates how the load-time for views (DLDB) versus triggers (OntoDB) appears unaffected. We used a non-logarithmic scale in Figure 1(b), to contrast how dramatically different DLDB and OntoDB perform, especially on chain queries (queries 10 and 13). Please see [21] for the full report.



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

Fig. 1. LUBM benchmark performance

2.3 Challenges

Therefore, the main challenges that we aim to address are:

- *Scalability.* Traditional knowledge base reasoners do not scale and disk-based solutions rarely take full advantage of intrinsic logical features of databases.
- *Expressiveness.* Database-based approaches that do use logical features, such as views, do not support negative knowledge, nor foreign-keys, and therefore do not readily detect inconsistencies.
- *Generalizability.* Specific modeling techniques make query translation and plan generation excessively complicated.

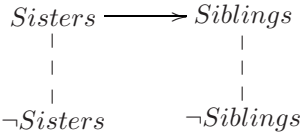
3 Our Solution

The key idea of using ontology databases to detect inconsistencies is to introduce explicit negations into the database. Typically, under the closed-world assumption (CWA), negative information is left out of the database and assumed false by default. Contradictions are easily detected by placing an integrity constraint on a relation and its negative counterpart, that is, $\neg(p \wedge \neg p)$. We call this an *exclusion dependency* because it ensures that p and $\neg p$ are mutually exclusive or disjoint. Horn rules with negations are implemented as triggers as usual. This way, as inferences are made, data is propagated forward such that if both $p(c)$ and $\neg p(c)$ are ever inferred (in any order), then the exclusion dependency will raise an inconsistency error.

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

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

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



We refer to these structures as *not-gadgets*. The not-gadget is the key to inconsistency detection using an ontology database.

Definition 1. *The* Ontology Database Language (ODL) *consists of the following grammar, where 'a' and 'b' are constants while 'x' and 'y' are variables:*

$$\begin{aligned} STMT &:= FACT \mid ATOM \mid RULE \\ FACT &:= true \mid false \mid C(a) \mid P(a,b) \\ ATOM &:= FACT \mid C(x) \mid P(x,y) \\ CONJ &:= ATOM \mid ATOM \wedge CONJ \\ RULE &:= CONJ \rightarrow ATOM \end{aligned}$$

Definition 2. *An* ontology database *using triggers (resp., views) implements the ODL language with typical first-order semantics such that: (i) ATOMs of the form C(x) or P(x,y) is a unary or binary table of the same predicate name, (ii) FACTs of the form C(a) or P(a,b) are tuples in the table of the same name, (iii) CONJs are SQL queries with join clauses unifying bound variables and where clauses grounding constants, and (iv) RULEs are triggers (resp., views) as described above.*

Definition 3. *A* conjunctive query *is an expression of the form* $\{\boldsymbol{x} | CONJ(\boldsymbol{x},\boldsymbol{y})\}$*, where the vector of variables,* $\boldsymbol{x}$*, are distinguished and meant to be bound and returned as answers and* $\boldsymbol{y}$ *are non-distinguished variables which are existentially qualified and for which the bindings can be discarded once the answer is computed.*

Claim. An ontology database is sound and complete[1] for conjunctive queries over an ODL ontology plus its extension, i.e., data.

Proof. (sketch) An ontology database is an implementation of the basic forward-chaining with *modus ponens* algorithm, where triggers implement *generalized modus ponens*, which is known to be sound and complete for Horn logic [21].

Clearly, an ontology database answers conjunctive queries in polynomial time with respect to the data because the extension is precomputed and we perform no reasoning at query-time. Therefore, computing answers takes only as long as the corresponding SQL query takes to run on the database.

[1] Recall from Section 2.2, by sound and complete we mean the usual logic formalism, i.e., everything derivable is entailed and vice-versa. Any answer returned by the database is considered derived by it.

Definition 4. *The* Ontology Database Language with negations (ODLn) *consists of the following grammar, where 'a' and 'b' are constants while 'x' and 'y' are variables:*

$$\begin{aligned} STMT &:= FACT \mid ATOM \mid RULE \\ FACT &:= true \mid false \mid C(a) \mid P(a,b) \mid \neg C(a) \mid \neg P(a,b) \\ ATOM &:= FACT \mid C(x) \mid P(x,y) \mid \neg C(x) \mid \neg P(x,y) \\ CONJ &:= ATOM \mid ATOM \wedge CONJ \\ RULE &:= CONJ \rightarrow ATOM \end{aligned}$$

Definition 5. *An* ontology database with negations *using triggers implements the ODLn language with typical first-order semantics in the same way as an ontology database would with the following additions: (i) for every ATOM of the form $\neg C(x)$ or $\neg P(x,y)$, the database includes a table of the same predicate name, prefixed with an underscore ("_") to represent "negation", (ii) for every pair of tables A and _A, we implement a special integrity constraint (hereafter referred to as an* exclusion dependency*) which prevents any tuple from appearing in both tables simultaneously, that is, we internally enforce $\neg(A \wedge \neg A)$.*

Definition 6. *A* not-gadget *is a database trigger implementing any rule of the form $A \rightarrow B$ where A or B can be either positive or negative ATOMs together with an exclusion dependency.*

Claim. An ontology database with negations is no longer complete for conjunctive queries over an ODLn ontology plus its extension.

Proof. (sketch) Generalized *modus ponens* is known to be incomplete in the presence of negations. We refer the reader to some counter examples such as *Andrea's Example* in [12].

3.1 Semantics of Deletion

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 [26]. 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 [10].

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.

Claim. 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, 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 structures.

3.2 Discussion

Unlike views, triggers cooperate well with integrity constraints in most database management systems. Sophisticated databases that maintain materialized views might be able to support foreign-keys referring to a view, but MySQL does not. Triggers do not have this problem: we can easily implement domain and range restrictions as foreign-key constraints. For the same reason, view-based approaches cannot be extended to include the idea of negations because MySQL cannot put an exclusion dependency on views. As further motivation, the event-driven modality of triggers holds bodes well for the distributed nature of the Semantic Web, since the event-driven model is one of the basic parallel programming design patterns – we envision distributed, ontology databases as the next step in our work.

On the other hand, using triggers comes with a significant potential drawback. Without provenance, truth maintenance is quite difficult in general. For example, we may need a mechanism to rollback *all* inferred knowledge, restructure based on an evolved ontology, then re-propagate facts. The naive approach, which would be to delete and start over, could be impractical if the ontology changes frequently. As we noted earlier, a binary property (is_inferred) would help with dropping inferred knowledge, but re-propagating is still difficult. In general, ontology evolution poses a number of truth maintenance problems which we do not address in work.

4 Case Study: The Serotonin Example

The goal of this case study is to illustrate a practical, real-world problem using the not-gadget to find inconsistencies. The serotonin example presented in Section 4.1 provides precisely the right motivation. In summary, we found 75 logical inconsistencies in the Gene Ontology plus annotations from ZFIN and MGI.

We discuss four possible explanations for these inconsistencies in Section 4.2. Finally, although not reported here in detail, we also conducted several other experiments on synthetic datasets[2] to confirm the soundness of our approach for detecting inconsistencies as described, yielding 100% precision and 100% recall. There were no significant differences in performance and scalability measures as reported in [21]. This case study illustrates our general design and methods.

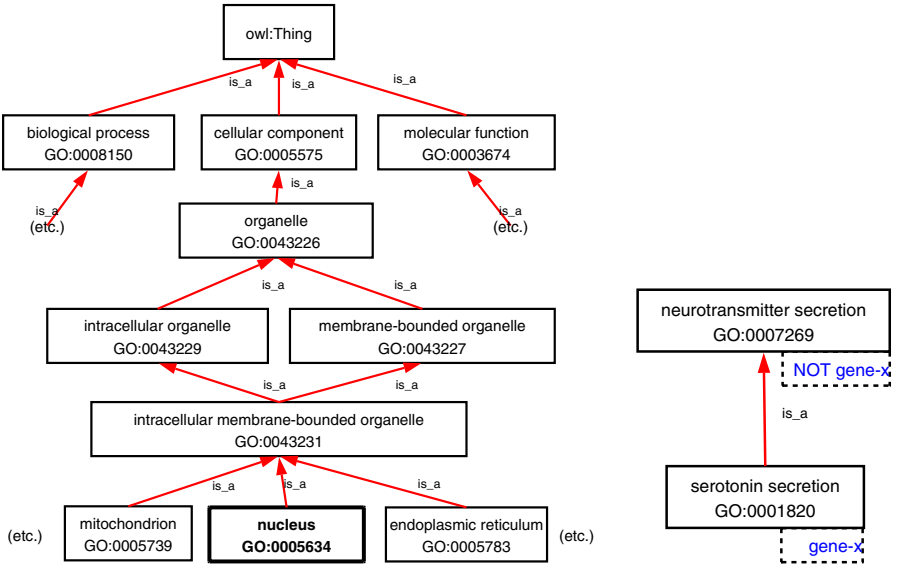
4.1 Motivation

The Gene Ontology and Annotations. The Gene Ontology (GO) [1,13] is used to specify the molecular functions, biological processes and cellular locations of gene products for the purpose of facilitating information sharing, data fusion and exchange [18] 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" (mereological) relationships, forming nearly 40,000 links among concepts[3]. Figure 2(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[4], 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 [17]. The problem is the number of gene annotations is several orders of magnitude beyond 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[5].

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[6]." But negative data are clearly in the minority. Compared to the 91,000 or so positive assertions from ZFIN, only 40 were negative facts; only 292 out of 154,000 facts from MGI were negative[7].

A negative annotation means a gene does not belong to the specified class within the context of a given experiment. As our results confirm in Section 4.2, 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[8]. However, in the same

[2] Varying ontology depth, breath, sparseness and data size and distribution.
[3] 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.
[4] Taken April 1, 2009.
[5] Recent reports as of April have annotations reaching over 40 million in number.
[6] `http://zfin.org/cgi-bin/webdriver?MIval=aa-markerview.apg&OID=ZDB-GENE-040718-342`
[7] Taken January, 24 2009.
[8] `http://zfin.org/cgi-bin/webdriver?MIval=aa-markergoview.apg&OID=ZDB-GENE-030319-2`



(a) The GO concept graph for nucleus. (b) The serotonin example.

Fig. 2. Examples from the Gene Ontology

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

The Serotonin Example. Hill *et al.* specify a concrete and illustrative example of a recently discovered flaw in the GO ontology, which we refer to as the *serotonin example* [18]. 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 inconsistencies in the ontology based on annotations from the model organism databases such as ZFIN and MGI.

Figure 2(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[9].

Of particular note, a logical inconsistency does not necessarily entail a type-type inconsistency! In fact, there are three possibilities for a given detected inconsistency: (1) the positive annotation is incorrect, (2) the negative annotation is incorrect, or (3) the subsumption relationship is incorrect. We may even allow a fourth possibility: (4) the inconsistency is admissible (i.e., an exception or anomaly). It so happened in the case of serotonin that biologists carefully investigated each of the possibilities to finally conclude that a type-type inconsistency was present. Therefore, each inconsistency raises new hypotheses that should be investigated further by biologists – some may lead to new biological insights, refined models, or improved automated techniques.

4.2 Experiment and Results

Setup and Design. An unremarkable laptop computer (1.8 GHz Centrino, 1GB of RAM, Windows XP) was used to process the GO ontology and generate the corresponding MySQL database schema representing the GO ontology database with negations. We used the OWL-API[10] and Java to implement this tool. An unremarkable desktop system (1.8 GHz Pentium, 512MB of RAM, Ubuntu Linux) was used as the MySQL database server.

We processed annotations from both the Zebrafish Information Network (ZFIN) [29] and the Mouse Genome Informatics (MGI) [5] databases taken on January 22, 2009 from the GO website[11]. Only is_a relationships were implemented. Our goal was to detect inconsistencies based on the ontology plus annotations. Detected inconsistencies were raised in an error log.

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;

[9] The GO might be characterized as medium-sized.
[10] `http://owlapi.sourceforge.net/`
[11] `http://www.geneontology.org/gene-associations/`

(4) load the annotations in to the MySQL database; (5) check the error log for the detected inconsistencies.

Results. The entire load-time (steps 1-4) takes under two hours to complete (30 minutes to load the schema [step 1 & 2] and 80 minutes to load the data [step 4]), consistent with previously reported performance results (see Figure 1(a)). We found 75 logic inconsistencies. Furthermore, to the best of our ability, in conjunction with a domain expert using the GOOSE[12] database, we confirmed that these results appear to maintain 100% precision and 100% recall. Admittedly, however, we have no gold-standard to make a firm judgement – if one existed, the serotonin example would not be an open problem!

The observed logic inconsistencies fell into three categories. We provide the following examples:

1. Intra-species logic inconsistencies between experimentally supported manual annotations: The zebrafish *p2rx2*[13] gene 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*[14] gene 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*[15], 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*[16] gene 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).

Discussion. 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 that conflict with experimentally supported manual annotations (e.g., inferred by direct assay or physical interaction), such as the *lzic* example above, suggest that the automated

[12] http://www.berkeleybop.org/goose

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

[14] http://zfin.org/cgi-bin/webdriver?MIval=aa-markergoview.apg&OID=ZDB-GENE-000616-1

[15] http://www.informatics.jax.org/javawi2/servlet/WIFetch?page=markerGO&key=33374

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

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 representation and reasoning would be necessary to resolve whether such cases were of biological interest or not.
4. Of the 75 inconsistencies discovered, none appeared to indicate a type-type error in the ontology, as was the case with serotonin, confirming that efforts by the GO consortium to maintain consistency in the ontology are well spent.

Of all the results, the intra-species inconsistencies arising from direct evidence versus automated electronic annotations were of particular significance and constitute an important biomedical informatics 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 regularly for evaluating electronic transfer rules. This specialized SQL query, which generates precisely the conflicts we discovered has been submitted to the Gene Ontology consortium[17].

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 [28], beyond the scope of this work.

5 Conclusion

We presented ontology databases with not-gadgets. The main problem we aimed to address is the poor scalability of reasoning over large numbers of instances, especially considering negative instances, by taking advantage of particular features of relational databases such as views, triggers and integrity constraints to implement not only subsumption-like reasoning, but particularly inconsistency detection. Unlike other approaches aimed at addressing scalability of RDF stores,

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

we do not use an external theorem prover to perform reasoning. Moreover, we also aimed to address expressiveness and generalizability of techniques by extending Horn-like logics with negations and by using a generic decomposition storage model.

We highlighted two important challenges and other future work. First, ontology databases using triggers to forward propagate knowledge are not resilient to changes in the ontology. Although this is a significant drawback of the approach, the time it takes to completely reload the entire ontology and its annotations makes it useable in practice. It can be completely reloaded overnight without interfering with daily activities. The speed gained during query answering is so significant that it warrants pre-computing the knowledge upfront. One future direction would be to tackle the ontology evolution problem by isolating those update events that can be performed on-line, versus those that require significant off-line restructuring. Second, we should consider the possibility of admissible contradictions or paraconsistency. Distinguishing between biological inconsistencies that are admissible, versus those that highlight important biological differences (and also admissible), versus those that are actually due to a logical error and non-admissible is an important challenge to consider for future theoretical work. Finally, a natural direction for this work would be to implement distributed ontology databases, where inferences are forward-propagated across networks of ontology databases using triggers with message-passing protocols. In essence, this would constitute an implementation of the heterogeneous data exchange model for information integration using an ontology database approach.

We used our technique toward the serotonin example, the problem of detecting inconsistencies between the GO and annotations curated in various model organism databases. We found 75 inconsistencies that prove to have important implications in biology, which include: (1) methods for refining transfer rules used for inferring electronic annotations, and (2) highlighting possible significant biological differences across species worth investigating. In this study, we only considered the is_a hierarchy of the GO. In the future, the current study could be expanded by taking into consideration all the transitive relationships in the GO, including annotation data from more species, and using more sophisticated computational methods (that do not rely on gene symbol string matches) for determining which genes should be considered to have conflicting annotations. The addition of these measures could increase the number of significant conflicts found.

Acknowledgements. We thank the ZFIN group, Zena M. Ariola, Gwen A. Frishkoff and Sarah Douglas for their feedback on and contributions to this work.

References

1. The Gene Ontology (GO) project in 2006. Nucleic Acids Research, 34(Database issue) (January 2006)
2. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: SW-Store: A Vertically Partitioned DBMS for Semantic Web Data Management. VLDB Journal 18(2), 385–406 (2009)

3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
4. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: International Semantic Web Conference, pp. 54–68 (2002)
5. Bult, C.J., Eppig, J.T., Kadin, J.A., Richardson, J.E., Blake, J.A.a.: The Mouse Genome Database (MGD): mouse biology and model systems. Nucleic Acids Research 36(Database issue) (January 2008)
6. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-Lite: Tractable Description Logics for Ontologies. In: AAAI 2005: Proceedings of the 20th National Conference on Artificial Intelligence, pp. 602–607 (2005)
7. Christophides, V., Karvounarakis, G., Plexousakis, D., Scholl, M., Tourtounis, S.: Optimizing taxonomic semantic web queries using labeling schemes. Journal of Web Sematics 1, 207–228 (2004)
8. Copeland, G.P., Khoshafian, S.N.: A decomposition storage model. In: SIGMOD 1985: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 268–279. ACM, New York (1985)
9. Curé, O., Squelbut, R.: A Database Trigger Strategy to Maintain Knowledge Bases Developed Via Data Migration. In: Bento, C., Cardoso, A., Dias, G. (eds.) EPIA 2005. LNCS (LNAI), vol. 3808, pp. 206–217. Springer, Heidelberg (2005)
10. Donini, F.M., Nardi, D., Rosati, R.: Description logics of minimal knowledge and negation as failure. ACM Trans. Comput. Logic 3(2), 177–225 (2002)
11. Dowek, G., Hardin, T., Kirchner, C.: Theorem Proving Modulo. Journal of Automated Reasoning 31, 2003 (1998)
12. Franconi, E.: Ontologies and databases: myths and challenges. In: Proceedings of the VLDB Endowment, vol. 1(2), pp. 1518–1519 (2008)
13. Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. Nature Genetics 25, 25–29 (2000)
14. Goble, C., Stevens, R.: State of the nation in data integration for bioinformatics. Journal of Biomedical Informatics (February 2008)
15. Guo, Y., Pan, Z., Heflin, J.: An Evaluation of Knowledge Base Systems for Large OWL Datasets. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 274–288. Springer, Heidelberg (2004)
16. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics 3(2-3), 158–182 (2005)
17. Haarslev, V., Möller, R.: High Performance Reasoning with Very Large Knowledge Bases: A Practical Case Study. In: IJCAI 2001: Proceedings of the International Joint Conferences on Artificial Intelligence, pp. 161–168 (2001)
18. Hill, D.P., Smith, B., McAndrews-Hill, M.S., Blake, J.A.: Gene Ontology annotations: what they mean and where they come from. BMC Bioinformatics 9(5) (2008)
19. Horrocks, I., Li, L., Turi, D., Bechhofer, S.: The Instance Store: DL Reasoning with Large Numbers of Individuals. Description Logics (2004)
20. Lee, J., Goodwin, R.: Ontology Management for Large-Scale E-Commerce Applications. In: DEEC 2005: Proceedings of the International Workshop on Data Engineering Issues in E-Commerce, pp. 7–15. IEEE Computer Society, Washington (2005)

21. LePendu, P., Dou, D., Frishkoff, G.A., Rong, J.: Ontology Database: a New Method for Semantic Modeling and an Application to Brainwave Data. In: Ludäscher, B., Mamoulis, N. (eds.) SSDBM 2008. LNCS, vol. 5069, pp. 313–330. Springer, Heidelberg (2008)
22. Motik, B., Horrocks, I., Sattler, U.: Bridging the Gap Between OWL and Relational Databases. In: WWW 2007: Proceedings of the 16th International Conference on World Wide Web, pp. 807–816 (2007)
23. Neumann, T., Weikum, G.: Scalable Join Processing on Very Large RDF Graphs. In: SIGMOD 2009: Proceedings of the ACM SIGMOD International Conference on Management of Data (to appear, 2009)
24. Reiter, R.: Deductive Question-Answering on Relational Data Bases. Logic and Data Bases, 149–177 (1977)
25. Reiter, R.: Towards a Logical Reconstruction of Relational Database Theory. In: Brodie, M.L., Mylopoulos, J., Schmidt, J.W. (eds.) On Conceptual Modelling—Perspectives from Artificial Intelligence, Databases, and Programming Languages. Topics in Information Systems, pp. 191–233. Springer, Heidelberg (1984)
26. Reiter, R.: What should a database know? In: PODS 1988: Proceedings of the seventh ACM Symposium on Principles of Database Systems, pp. 302–304. ACM, New York (1988)
27. Rubin, D.L., Musen, M.A., et al.: National Center for Biomedical Ontology: Advancing Biomedicine through Structured Organization of Scientific Knowledge. OMICS: A Journal of Integrative Biology 10(2), 185–198 (2009)
28. Slaney, J.K.: Relevant Logic and Paraconsistency. Inconsistency Tolerance, 270–293 (2005)
29. Sprague, J., Westerfield, M., et al.: The Zebrafish Information Network: the zebrafish model organism database provides expanded support for genotypes and phenotypes. Nucleic Acids Research 36, D768–D772 (2007)
30. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, vol. I. Computer Science Press (1988)
31. Vasilecas, O., Bugaite, D.: An algorithm for the automatic transformation of ontology axioms into a rule model. In: CompSysTech 2007: Proceedings of the International Conference on Computer Systems and Technologies, pp. 1–6. ACM, New York (2007)
32. Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S.: Ontology-based integration of information — a survey of existing approaches. In: Stuckenschmidt, H. (ed.) IJCAI 2001: Workshop on Ontologies and Information Sharing, pp. 108–117 (2001)