Efficient Semantic Querying of Relational Databases with Resolution

Alexandre Riazanov

RuleML

alexandre.riazanov@gmail.com

Abstract. We address the problem of semantic querying of relational databases (RDB) modulo knowledge bases using very expressive knowledge representation formalisms, such as full first-order logic or its various fragments. We propose to use a first-order logic (FOL) reasoner for computing *schematic answers* to deductive queries, with the *subsequent instantiation* of these schematic answers using a conventional relational DBMS. In this paper, we outline the main idea of this technique – using *abstractions of databases* and *constrained clauses* for deriving schematic answers. The proposed method can be directly used with *regular RDB*, including *legacy databases*. Moreover, we propose it as a potential basis for an efficient Web-scale semantic search technology.

1 Introduction.

1.1 Settings and motivation.

Consider the following scenario. Suppose we have one[1] relational database (RDB), one or more expressive knowledge bases (KB) for domains to which the data in the RDB is related (e. g., rule bases in expressive sublanguages of RuleML [7, 2] and/or ontologies in OWL). We would like to work with *arbitrary* (reasonably well designed) RDBs, and, consequently, the database relations are not assumed to directly correspond to relations described by the KBs. So, optionally, we may also have some mapping between the RDB schema and the logical language of the domains, i. e., a logical description of the relations in the RDB, to link them to the concepts and relations defined by the KBs. In these settings, we would like to be able to formulate queries logically and answer them w. r. t. the KBs and the RDB treated virtually as a collection of ground atomic facts (e. g., by viewing each table row as a separate ground fact). *To make this process efficient, we would like to use the modern RDB technology as much as possible by delegating as much work as possible to the RDBMS hosting the database*.

We propose a method to implement this scenario, based on the use of *resolution for incremental transformation of semantic queries into sequences of SQL queries* that can be directly evaluated on the RDB, and whose results provide answers to the original queries.

We envisage two main applications for the proposed technology.

[1] In principle, our approach can be extended to multiple heterogeneous and distributed databases, but in this paper we assume, for simplicity, that we are dealing with just one DB.

Enhancing the interface to conventional relational databases. Flexible querying of conventional RDBs by non-programmer users is very problematic because real-life enterprise databases often have complex designs. Writing a correct query requires good understanding of technical details of the DB schema, such as table and attribute names, foreign key relationships, nullable fields, etc. So most of RDB querying by non-programmer users is done with preprogrammed parameterised queries, usually represented as forms of various kinds.

Even when special methodologies are used, like Query-by-Example (see, e. g. [25]), that allow to hide some of the complexities of SQL and database designs from the end users, one important inherent limitation remains in force. Whereas mapping some domain concepts to the RDB schema elements may be easy, many other concepts may be much more difficult to map. For example, it is easy to select instances of the concept "student" if there is a table explicitly storing all students, but if the user wants to extract a list of all members of a department in a university, he may have to *separately* query different tables storing information about students, faculty and support staff (assuming that there is no table specifically storing members of all these kinds), and then create a union of the results.

This example exposes well the root of the problem: mapping some domain concepts to the data is difficult because it requires *application of the domain knowledge*. In the example, the involved piece of domain knowledge is the fact that students, faculty and support staff are all department members, and the user has to apply it manually to obtain the required results.

Semantic querying is based on automatic application of domain knowledge formalised in the form of, e. g., rules and ontological axioms. In this approach, DB programmers "semantically document" their DB designs by providing an explicit mapping between the RDB schemas and domain terminologies, e. g., in the form of logical axioms. This alone allows an end user to formulate queries directly in the terminology of the domain, without even a slightest idea about how the underlying RDBs are structured[2]. However, the biggest advantage comes from the fact that reasoning w. r. t. additional, completely external KBs can be employed to generate and justify some answers, which makes querying not just *semantic*, as in [28], but also *deductive*. In our current example, the user can provide, as a part of the query, some KB that links the relations of being a department member, being a student in the department, etc. In some application contexts, it is important to be able to use rather expressive KBs for such purposes. Rule-based KBs and expressive DL ontologies are of a special interest, especially in combination.

Web-scale semantic search. The Semantic Web is accumulating a lot of data in the form of RDF and OWL assertions referring to various formalised vocabularies – ontologies. In some cases the expressivity of RDF(S) and OWL may not be enough[3]

---

[2] This does not alleviate the need for convenient query interfaces, but they are outside the scope of this paper.

[3] For example, OWL cannot express the following simple rule $hasUncle(X, Y)\ :-\ hasParent(X, Z),\ hasBrother(Z, Y)$ [3]. OWL also restricts the arity of predicates to 2 and does not directly support functions, thus limiting knowledge engineering possibilities. More detailed discussion of this issue is outside the scope of this paper.

and knowledge bases in other formalisms, e. g., RuleML [7, 2], RIF [1] or SWRL [3], have to be used to capture more complex dependencies between domain concepts and relations, thus making the data descriptions sufficiently semantically rich.

The utility of the Semantic Web data will strongly depend on how easily and how efficiently users and agents can query it. Roughly speaking, we need to *query extremely large volumes of highly distributed data modulo expressive knowledge bases*, so that not only direct answers based on the stored data are returned, but also implied answers that can only be obtained by reasoning.

The approach proposed here may be a part of a solution to this problem: large sets of RDF triples and OWL data descriptions (coming from Semantic Web documents) can be loaded into a relational database and then queried deductively modulo the relevant knowledge bases. Different DB layouts can be used, depending on the nature of the data being loaded. For example, if we load an OWL ABox, we can have a separate one-column table for keeping instances of each class and, similarly, a separate two-column table for keeping assertions of each property[4]. Loading data descriptions into an RDB is a linear operation, so it is unlikely to become a real performance bottleneck. Moreover, we can start producing answers even before the data is fully loaded. So the efficiency of such a scheme depends mostly on how efficiently the deductive querying on the RDB can be done.

Just like text-based Web search engines do not indiscriminately scan all the accessible documents each time a new query is processed, semantic search systems cannot examine all accessible data descriptions in every retrieval attempt. Instead, some form of indexing is necessary that would allow to avoid downloading data that is irrelevant to a specific query, and would focus the processing on the sets of assertions that are likely to contribute to some answers to the query. We will show that the core feature of our approach to deductive querying of RDB – incremental query rewriting – suggests a natural way of semantically indexing distributed data sources.

1.2 Outline of the proposed method.

To implement the target scenario, we propose to use a first-order logic reasoner in combination with a conventional RDBMS, so that the reasoner does the “smart” part of the job, and the RDBMS is used for what it is best at – relatively simple processing of large volumes of relational data by computing table joins. Roughly, the reasoner works as a query preprocessor. It accepts a semantic query, the relevant knowledge bases and a semantic mapping for a DB as its input, and generates a (possibly infinite) number of expressions which we call *schematic answers*[5], that can be easily converted into SQL queries. These SQL queries are then evaluated on the DB with the help of the RDBMS. The union of the results for these SQL queries contains all answers to the original deductive query.

This idea can be implemented with a relatively simple architecture as shown in Figure 1. The architecture introduces two main modules – a reasoner for finding schematic

[4] These is the scheme used in all examples throughout the paper.

[5] In earlier versions of this paper we used the term *generic answers*, which clashes with the classification proposed in [9].

solutions and an SQL generator to turn these solutions into SQL queries. We also assume that some off-the-shelf RDBMS is used to answer the SQL queries. All three components (can) work in parallel: while the reasoner searches for another schematic answer, the SQL generator can process some previous general solutions and the RDBMS can generate instances for some earlier general solutions and communicate them to the user.

Optionally, the reasoner may try to prune the search space by checking certain constraints over the RDB (details will be provided in Section 4). These constraints are also converted into SQL queries and sent to the RDBMS for evaluation. The results of the evaluation ($'satisfiable'$ or $'unsatisfiable'$) are sent back to the reasoner which can use the absence of solutions for a constraint as a justification for suppressing certain inferences.

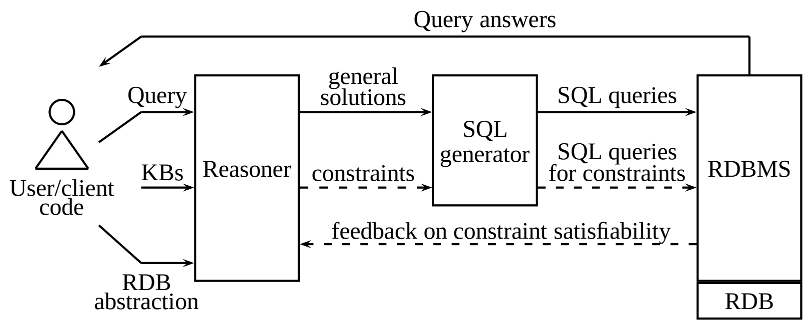

Fig. 1. Architecture for deductive query answering

The rest of this paper is structured as follows. In Section 2 we introduce the method intuitively. In Section 3 we provide a minimal mathematical justification of usability of our approach by demonstrating soundness and completeness of some standard resolution-based calculi for rewriting semantic queries into sequences of schematic answers. In Section 4 we describe one optimisation specific to schematic answer search. In Section 6 we briefly discuss how semantic indexing can be done using data abstractions, in the context of Web-scale retrieval. In Section 5 we provide an algorithm for converting the logical representation of schematic answers into SQL. Finally, Sections 7 and 8 briefly describe some related and future work.

2 Informal method description.

We *model* an RDB as a finite set of ground atomic formulas, so that RDB table names provide the predicates, and rows are conceptually treated as applications of the predicates to the row elements. In the example below, we have a table $takesCourse$ from a University DB, keeping information about which student takes which course, whose rows are mapped to a set of facts.

takesCourse	**student**	**course**		
	s1	c1	$\longrightarrow$	takesCourse(s1,c1)
	s2	c2	$\longrightarrow$	takesCourse(s2,c2)
	s3	c3	$\longrightarrow$	takesCourse(s3,c3)
	...	...	...	

Before we proceed with more important things, note that in all our examples in this paper, the data is assumed to be a relational representation of some DL ABoxes. This is done not to clutter the presentation of the main ideas with RDB schema-related details. In particular, there is no need for a special RDB-to-KB mapping because the RDB tables directly correspond to concepts and properties. It bears repeating that this assumption is made *only to simplify the presentation* – our approach is applicable to any RDBs, including legacy ones, as long as their design allows reasonable semantic mapping.

Now, suppose we are trying to answer a query over our RDB deductively, e. g., modulo some KB.

Naive approach as a starting point. Hypothetically, we can explicitly *represent the DB as a collection of ground atomic facts* and use some resolution-based FOL reasoner supporting query answering, e.g., Vampire [27] or Gandalf [30].

Even if we have enough memory to load the facts, this approach is likely to be very inefficient for the following reason. If the RDB is large and the selectivity of the query is not very high, we can expect that *many answers will be obtained with structurally identical proofs*. For example, if our DB contains facts $graduateStudent(s_1), \ldots,$ $graduateStudent(s_{100})$ (representing some table $graduateStudent$ which simply keeps a list of all graduate students), the facts will give rise to 100 answers to the query $student(X)$[6], each having a refutational proof of the form shown in Figure 2 (where $grStud$, $takesC$, $pers$ and $stud$ abbreviate $graduateStudent$, $takesCourse$, $person$ and $student$, and $sk0$ is a Skolem function).

This example is intended to demonstrate how *wasteful reasoning on the per-answer basis* is. Roughly speaking, the required amount of reasoning is multiplied with the number of answers. Even if the selectivity of the query is very high, the reasoner is still likely to waste a lot of work in unsuccessful attempts represented by derivations not leading to any answers.

Note that these observations are not too specific to the choice of the reasoning method. For example, if we used Prolog or a tableaux-based DL reasoner, we would have a similar picture: the same rule applications would be performed for each answer s_i.

Main idea. The main idea of our proposal is that *answers with similar proofs should be obtained in bulk*. More specifically, we propose to *use reasoning to find schematic answers* to queries, which can be later very efficiently *instantiated by querying the RDB via the standard highly optimised RDBMS mechanisms*. Technically, we propose to search for the schematic answers by *reasoning on an abstraction of the RDB in some resolution- and paramodulation-based calculus* (see [5, 21]). The abstraction and

[6] Query 6 from LUBM [16].

[0]	$\neg grCourse(X) \vee course(X)$	; input, $grCourse \sqsubseteq course$
[1]	$grStud(s_i)$	; input, DB row
[2]	$\neg grStud(X) \vee grCourse(sk0(X))$	; input, from $grStud \sqsubseteq \exists takesC.grCourse$
[3]	$grCourse(sk0(s_i))$	; from [1] and [2]
[4]	$course(sk0(s_i))$	; from [0] and [3]
[5]	$\neg grStud(X) \vee takesC(X, sk0(X))$	; input, from $grStud \sqsubseteq \exists takesC.grCourse$
[6]	$takesC(s_i, sk0(s_i))$	; from [1] and [5]
[7]	$\neg takesC(X, Y) \vee \neg course(Y) \vee$ $\neg pers(X) \vee stud(X)$	; input, from $stud \equiv pers \sqcap \exists takesC.course$
[8]	$\neg course(sk0(s_i)) \vee \neg pers(s_i) \vee stud(s_i)$	; from [6] and [7]
[9]	$\neg pers(s_i) \vee stud(s_i)$	; from [4] and [8]
[10]	$\neg grStud(X) \vee pers(X)$	; input, $grStud \sqsubseteq pers$
[11]	$pers(s_i)$	; from [1] and [10]
[12]	$stud(s_i)$	; from [9] and [11]
[13]	$\neg stud(X) \vee answer(X)$	; input, query $find\ X.stud(X)$
[14]	$answer(s_i)$	; from [12] and [13]

Fig. 2. Resoluton derivation of the answer $X := s_i$ for the query $stud(X)$.

the reasoning on the abstraction should be organised in such a way that the obtained schematic answers can be turned into *regular RDBMS queries* (e.g., SQL queries).

Constrained clauses and table abstractions. To illustrate our main idea, we apply it to the current example. The clause $grStud(X) \mid grStud(X)$ is the *abstraction* of the relevant part of the RDB, i.e., it represents (generalises) all the facts $grStud(s_1), \ldots,$ $grStud(s_{100})$. This is a very important feature of our approach, so we emphasise that a potentially very large set of facts is compactly represented with just one clause. The part before "|" is the ordinary logical content of the clause. What comes after "|" is a special constraint. These constraints will be *inherited* in all inference rules, *instantiated* with the corresponding unifiers and *combined* when they come from different premises, just like, e. g., ordering or unifiability constraints in paramodulation-based theorem proving [21]. Although our constraints can be used as regular constraints – that is to identify redundant inferences by checking the satisfiability of the associated constraints w.r.t. the RDB (see Section 4) – *their main purpose is to record which RDB fact abstractions contribute to a schematic answer and what conditions on the variables of the abstractions have to be checked when the schematic answer is instantiated, so that the obtained concrete answers are sound*.

A derivation of a schematic answer for the query $student(X)$, covering all the concrete solutions $X := s_1, \ldots, X := s_{100}$, is shown in Figure 3. Note that the last inference simply merges three identical atomic constraints. Also note that we write the answer literals on the constraint sides of the clauses, because they are not intended for resolution.

SQL generation. Semantically the derived schematic answer $\square \mid \neg answer(X),$ $grStud(X)$ means that if some value x is in the table $graduateStudent$, then x is a legitimate concrete answer to the query. So, assuming that **id** is the (only) attribute in the RDB table representing the instances of $graduateStudent$, the derived schematic

[0]	$\neg grCourse(X) \vee course(X)$	; input, KB
[1]	$grStud(X) \mid grStud(X)$	; DB table abstraction
[2]	$\neg grStud(X) \vee grCourse(sk0(X))$	; input, KB
[3]	$grCourse(sk0(X)) \mid grStud(X)$	; from [1] and [2]
[4]	$course(sk0(X)) \mid grStud(X)$	; from [0] and [3]
[5]	$\neg grStud(X) \vee takesC(X, sk0(X))$	; input, KB
[6]	$takesC(X, sk0(X)) \mid grStud(X)$	; from [1] and [5]
[7]	$\neg takesC(X,Y) \vee \neg course(Y) \vee \neg pers(X) \vee stud(X)$	; input, KB
[8]	$\neg course(sk0(X)) \vee \neg pers(X) \vee stud(X) \mid grStud(X)$	; from [6] and [7]
[9]	$\neg pers(X) \vee stud(X) \mid grStud(X),\ grStud(X)$	; from [4] and [8]
[10]	$\neg grStud(X) \vee pers(X)$	; input, KB
[11]	$pers(X) \mid grStud(X)$	; from [1] and [10]
[12]	$stud(X) \mid grStud(X),\ grStud(X),\ grStud(X)$	; from [9] and [11]
[13]	$\neg stud(X) \mid \neg answer(X)$	; query
[14]	$\Box \mid \neg answer(X),\ grStud(X),\ grStud(X),\ grStud(X)$	; from [12] and [13]
[15]	$\Box \mid \neg answer(X),\ grStud(X)$	; from [14]

Fig. 3. Resolution derivation of some schematic answer for $stud(X)$.

answer $\Box \mid \neg answer(X), grStud(X)$ can be turned into the following simple SQL query:

SELECT **id** AS X
FROM graduateStudent

Evaluating this query over the RDB will return all the answers $X := s_1, \ldots, X := s_{100}$.

Resolution reasoning on a DB abstraction may give rise to *more than one schematic answer*. For example, $\Box \mid \neg answer(X), grStud(X)$ does not necessarily cover all possible solutions of the initial query – it only enumerates graduate students. If our KB also postulates that any person taking a course is a student, we want to select all such people as well. So, suppose that our DB also contains the facts $person(P_1), \ldots, person(P_{100})$, $takesCourse(P_1, C_1), \ldots, takesCourse(P_{100}, C_{100})$ and $course(C_1), \ldots,$ $course(C_{100})$ in the corresponding tables $person$, $takesCourse$ and $course$. These relations can be represented with the abstraction clauses $person(X) \mid person(X)$, $takesCourse(X,Y) \mid takesCourse(X,Y)$ and $course(X) \mid course(X)$. Simple reasoning with these clauses modulo, say, a KB containing the rule $student(P) :- person(P),\ takesCourse(P,C),\ course(C)$ or the DL axiom $person \sqcap \exists takesC.course \sqsubseteq student$, produces the schematic answer $\Box \mid \neg answer(X), person(X), takesCourse(X,Y), course(Y)$. Semantically it means that if table $takesCourse$ contains a record $\{\mathbf{student} = s, \mathbf{course} = c\}$, and tables $person$ and $course$ contain s and c correspondingly, then $X := s$ is a legitimate concrete answer. Thus, the schematic answer can be turned into the following SQL query:

```
SELECT person.id AS X
FROM person, takesCourse, course
WHERE person.id = takesCourse.student
   AND course.id = takesCourse.course
```

The join conditions $person.\mathbf{id} = takesCourse.\mathbf{student}$ and $course.\mathbf{id} = takesCourse.\mathbf{course}$ reflect the fact that the corresponding arguments of the predicates in the constraint attached to the schematic answer are equal: e.g., the only argument of $person$, corresponding to $person.\mathbf{id}$, and the first argument of $takesCourse$, corresponding to $takesCourse.\mathbf{student}$, are both the same variable X.

Incremental query rewriting. In general, resolution over DB abstractions in the form of constrained clauses may produce many, even infinitely many, schematic answers and, consequently, SQL queries. They are produced one by one, and the union of their answers covers the whole set of concrete answers to the query. If there is only a finite number of concrete answers, e. g., if the query allows concrete answers to contain only plain data items from the database, then all concrete answers are covered after some finite number of steps. In a sense, the original semantic query is rewritten as a sequence of SQL queries, so we call our technique *incremental query rewriting*.

Benefits. The main advantage of the proposed scheme is the *expressivity scalability*. For example, in applications not requiring termination, the expressivity of the knowledge representation formalisms is only limited by the expressivity of the full FOL[7], although specialised treatment of various FOL fragments is likely to be essential for good performance. The use of such a powerful logic as FOL as the common platform also allows easy practical simultaneous use of heterogeneous knowledge bases, at least for some data retrieval tasks. In particular, it means that users can freely mix all kinds of OWL and RDFS ontologies with all kinds of (first-order, monotonic) declarative rule sets, e. g., in RuleML or SWRL.

It is important that we don't pay too high a price in terms of performance, for the extra expressivity. The method has good data scalability: roughly, *the cost of reasoning is not multiplied by the volume of data*. Note also that we don't have to do any static conversion of the data into a different data model, e. g., RDF triples or OWL ABox – querying can be done on live databases via the hosting RDBMSs. All this makes our method potentially usable with very large databases in real-life settings.

An additional advantage of our approach is that answers to semantic queries can be relatively easily given rigorous explanations. Roughly speaking, if we need to explain a concrete answer, we simply instantiate the derivation of the corresponding schematic answer by replacing DB table abstractions with concrete DB rows, and propagating this data through the derivation. Thus, we obtain a resolution proof of the answer, which can be relatively easily analysed or transformed into a more intuitive representation.

3 Soundness and completeness of schematic answer computation.

So far we have only speculated that schematic answer search can be implemented based on resolution. In this section we are going to put it on a formal basis. We will show that in the context of FOL without equality some popular resolution-based methods can deliver the desired results. In particular, we will characterise a class of resolution-based calculi that are both sound and complete for query answering over database abstractions.

We assume familiarity of the reader with the standard notions of first-order logic, such as terms, formulas, literals and clauses, substitutions, etc., and some key results,

[7] Complete methods for efficient schematic answer finding in FOL *with equality* are yet to be formulated and proved formally (see the brief discussion in Section 8).

such as the Herbrand's theorem. Bibliographic references are provided for more specialised concepts and facts.

Deductive queries. In our settings, a *deductive query* is a triple $\langle DB, KB, \varphi\rangle$, where (i) the logical representation DB of some relational database is a set of ground atomic non-equality formulas, each representing a row in a table in the database, (ii) the *knowledge base* KB is a finite set of FOL axioms, corresponding to both the domain ontologies and semantic RDB schema mappings in our scenario, and (iii) the *goal* φ of the query is a construct of the form $\langle X_1, \ldots, X_k\rangle\langle Y_1, \ldots, Y_m\rangle C$, where C is a nonempty clause, $k, m \geq 0$, $\{X_1, \ldots, X_k, Y_1, \ldots, Y_m\} = vars(C)$, all X_i and Y_i are pairwise distinct. We call X_i *distinguished variables*, and Y_j *undistinguished variables* of the query. Intuitively, the deductive query represents a request to find all X_i, such that there exist some Y_j, such that $\varphi(\overline{X}, \overline{Y})$ is *inconsistent* with $DB \cup KB$. In other words, answers to the query refute φ rather than prove it[8]. This convention is made for technical convenience. Users of our technology can work in terms of positive queries.

Recording literals. In our settings, a clause with *recording literals*[9] is a construct of the following form: $C \mid \gamma$, where C is a regular first-order clause, possibly empty, and γ is a finite multiset of literals, possibly empty. We will say that the literals of γ are *recording literals*.

Semantically, $C \mid \lambda_1, \ldots, \lambda_n$ is the same as the regular clause $C \vee \overline{\lambda_1} \vee \ldots \vee \overline{\lambda_n}$, which will be denoted as $Sem(C \mid \lambda_1, \ldots, \lambda_n)$. All semantic relations between $Sem(C \mid \gamma)$ and other formulas are transferred to $C \mid \gamma$. For example, when we say that $C \mid \gamma$ is implied by something, it means that $Sem(C \mid \gamma)$ is implied, and vice versa.

Regular clauses will be often identified with clauses with empty recording parts, i.e., we will not distinguish C from $C \mid \emptyset$.

We say that a clause $C' \mid \gamma'$ subsumes the clause $C \mid \gamma$ iff there is a substitution θ that makes $C'\theta$ a submultiset of C, and $\gamma'\theta$ a submultiset of γ'. In this case we will also say that $C' \mid \gamma'$ is a *generalisation* of $C \mid \gamma$.

Concrete and schematic answers. We distinguish a special predicate symbol @[10]. A ground atomic formula $@(t_1, \ldots, t_k)$ is a *concrete* answer to the deductive query $\langle DB, KB, \langle X_1, \ldots, X_k\rangle\langle Y_1, \ldots, Y_m\rangle C\rangle$, if the clause $C[X_1/t_1, \ldots, X_k/t_k]$ is *inconsistent* with $DB \cup KB$ or, equivalently, the formula $\exists Y_1 \ldots Y_m \neg C[X_1/t_1, \ldots, X_k/t_k]$ is implied by $DB \cup KB$.

We say that a clause $\square \mid \gamma$ is a *schematic answer* to a deductive query $\langle DB, KB, \langle X_1, \ldots, X_k\rangle\langle Y_1, \ldots, Y_m\rangle C\rangle$, if every atomic ground formula of the form $@(t_1, \ldots, t_k)$ implied by $DB \cup \{\square \mid \gamma\}$, is a concrete answer to the query. Every such concrete answer will be called an *instance* of the schematic answer. For example, $@(s_1), \ldots, @(s_{100})$ are instances of the schematic answer $\square \mid \neg @(X),\ grStud(X)$ in the main example in Section 2.

Database abstractions. In our settings, a finite set DB' of clauses of the form $p(t_1, \ldots, t_k) \mid p(t_1, \ldots, t_k)$ is an *abstraction* of the logical representation DB of a

[8] Recall the part $\neg stud(X)$ of clause [13] from Fig. 2.

[9] We prefer this to the more general term "constrained clause" because we want to emphasise the nature and the role of our constraints, and to avoid confusion with other kinds of constraints used in automated reasoning and logic programming.

[10] Corresponds to the predicate $answer$ used in our previous examples.

database if for every atomic formula $\rho \in DB$, there is a clause $\rho' \mid \rho' \in DB'$ and a substitution θ, such that $\rho'\theta = \rho$. Note that *semantically* all clauses in DB' are tautologies, because $Sem(p(t_1, \ldots, t_k) \mid p(t_1, \ldots, t_k)) \;=\; p(t_1, \ldots, t_k) \vee \neg p(t_1, \ldots, t_k)$.

The simplest kind of an abstraction for an RDB is the set of all clauses $p(X_1, \ldots, X_k) \mid p(X_1, \ldots, X_k)$, where all X_i are pairwise distinct variables, and each p corresponds to a table in the RDB (see, e. g., clause [1] in Fig. 2). Dealing with such an abstraction can be viewed as reasoning on the schema of the RDB. However, in principle, we can have more specific abstractions. For example, if we know that the first column of our RDB table p contains only values a and b, we may choose to have two abstraction clauses: $p(a, X_2, \ldots, X_k) \mid p(a, X_2, \ldots, X_k)$ and $p(b, X_2, \ldots, X_k) \mid p(b, X_2, \ldots, X_k)$[11].

Calculi. In this paper we only deal with calculi that are sound and complete variants of resolution[12] (see, e. g., [5]). All inference rules in these calculi are of the form

$$\frac{C_1 \quad C_2 \quad \ldots \quad C_n}{D}$$

where C_i and D are ordinary clauses, and $n \geq 1$. Most such rules have a substitution θ associated with them, which is required to unify some subexpressions in C_i, usually atoms of complementary literals. Rules in the calculi that are of interest to us can be easily extended to clauses with recording literals as shown in Figure 4(a). So, for example, the binary resolution rule extended to clauses with recording literals is shown in Figure 4(b).

$$\frac{C_1 \mid \gamma_1 \quad C_2 \mid \gamma_2 \quad \ldots \quad C_n \mid \gamma_n}{D \mid \gamma_1\theta, \gamma_2\theta, \ldots, \gamma_n\theta}\ (a) \qquad \frac{C_1' \vee A \mid \gamma_1 \quad\quad C_2' \vee \neg B \mid \gamma_2}{C_1'\theta \vee C_2'\theta \mid \gamma_1\theta, \gamma_2\theta}\ (b)$$

$$\text{where } \theta = mgu(A, B)$$

Fig. 4. Inferences on clauses with recording literals: (a) general form, (b) binary resolution

If a calculus R' is obtained by extending the rules of a calculus R to clauses with recording literals, we will simply say that R' is a *calculus with recording literals* and R is its *projection to regular clauses*.

Apart from nonredundant inferences, resolution calculi used in practice usually include some *admissible* redundant inferences. Implementers have the freedom of performing or not performing such inferences without affecting the completeness of the reasoning process. However, for the purposes of this paper it is convenient to assume that calculi being considered only contain nonredundant inferences. This assumption does not affect generality.

A calculus with recording literals is *sound* if Sem of the conclusion of every derivation is logically implied by the Sem images of the clauses in the leaves. It is obvious that a calculus with recording literals is sound if its projection to regular clauses is

[11] Moreover, we can have just one abstraction clause, e. g., $p(X_1, \ldots, X_k) \mid p(X_1, \ldots, X_k),\ X_1 \in \{a, b\}$ with the additional *ad hoc constraint* $X_1 \in \{a, b\}$, but this kind of optimisations is outside the scope of this paper.

[12] Paramodulation is also briefly discussed as a future research opportunity in Section 8.

sound because recording literals are fully inherited. A calculus with recording literals is *refutationally complete* if its projection to regular clauses is refutationally complete, i.e., an empty clause can be derived from any unsatisfiable set of clauses.

In this paper we will mention *fully specified calculi* to distinguish them from generic (parameterised) calculi. For example, the ordered binary resolution in general is not fully specified – it is a generic calculus *parameterised* by an order on literals. If we fix this parameter by specifying a concrete order, we obtain a fully specified calculus. We view a fully specified calculus as the set of all its elementary inferences.

We say that a fully specified calculus R with recording literals is *generalisation-tolerant* if every inference in R is generalisation-tolerant. An elementary inference

$$\frac{C_1 \mid \gamma_1 \quad C_2 \mid \gamma_2 \quad \ldots \quad C_n \mid \gamma_n}{D \mid \delta}$$

from the calculus R is generalisation-tolerant if for every generalisation $C'_i \mid \gamma'_i$ of a premise $C_i \mid \gamma_i$, the calculus R also contains an elementary inference of some generalisation $D' \mid \delta'$ of $D \mid \delta$, where the premises are a submultiset of $\{C_1 \mid \gamma_1,\ \ldots,\ C_{i-1} \mid \gamma_{i-1},\ \ C'_i \mid \gamma'_i,\ \ C_{i+1} \mid \gamma_{i+1},\ \ldots,\ C_n \mid \gamma_n\}$.

Unordered binary resolution and hyperresolution provide simple examples of generalisation-tolerant calculi. Their ordered versions using admissible orderings (see, e. g., [5]) also cause no problems because application of generalisation to a clause cannot make a maximal literal nonmaximal, because of the *substitution property* of admissible orderings: $L_1 > L_2$ implies $L_1\theta > L_2\theta$. Adding (negative) literal selection (see, e. g., [5]) requires some care. In general, if a literal is selected in a clause, its image, if it exists, in any generalisation should be selected too. Such selection functions are still possible. For example, we can select *all* negative literals that are maximal w. r. t. some ordering satisfying the substitution property. In this case, however, we can no longer restrict ourselves to selecting a single literal in a clause, because the ordering can only be partial.

Note that such calculi are the main working horses in several efficient FOL reasoners, e. g., Vampire.

Theorem 1 (soundness). Suppose R is a sound fully specified calculus with recording literals. Consider a deductive query $Q = \langle DB, KB, \langle X_1, \ldots, X_k\rangle\langle Y_1, \ldots, Y_m\rangle C\rangle$. Suppose DB' is an abstraction of DB. Suppose we can derive in R a clause $\Box \mid \gamma$ from $DB' \cup KB \cup \{C \mid \neg@(X_1, \ldots, X_k)\}$. Then $\Box \mid \gamma$ is a schematic answer to Q.

This is easy to prove by using the fact that all abstraction clauses are semantically tautologies. Details can be found in [26].

Theorem 2 (completeness). Suppose R is a refutationally complete and generalisation-tolerant fully specified calculus with recording literals. Consider a deductive query $Q = \langle DB, KB, \langle X_1, \ldots, X_k\rangle\langle Y_1, \ldots, Y_m\rangle C\rangle$. Suppose DB' is an abstraction of DB. Then, for every concrete answer $@(t_1, \ldots, t_k)$ to Q one can derive in R from $DB' \cup KB \cup \{C \mid \neg@(X_1, \ldots, X_k)\}$ a clause $\Box \mid \gamma$, such that $@(t_1, \ldots, t_k)$ is an instance of the schematic answer $\Box \mid \gamma$.

Proof. The refutational completeness of R means that we can construct a refutation Δ of $DB \cup KB \cup C[X_1/t_1, \ldots, X_k/t_k]$. The main idea of this proof is that in a generalisation-tolerant calculus finding an answer to a query is not much more difficult than just proving the answer. Technically, we will convert Δ into a derivation of a schematic answer covering the concrete answer $@(t_1, \ldots, t_k)$.

Assume that ρ_i, $i \in [1 \ldots p]$, are all the facts from DB that contribute to Δ (as leaves of the refutation). We can convert Δ into a derivation Δ' of a clause of the form $\square \mid \rho_1, \ldots, \rho_p, \neg A_1, \ldots, \neg A_n$, where $p, n \geq 0$ and all atoms $A_i = @(t_1, \ldots, t_k)$, from the clauses $\rho_1 \mid \rho_1, \quad \ldots, \quad \rho_p \mid \rho_m$, $C[X_1/t_1, \ldots, X_k/t_k] \mid \neg @(t_1, \ldots, t_k)$ and some clauses from KB. To this end, we simply add the recording literals in the corresponding leaves of Δ and propagate them all the way to the root. Obviously, $DB \cup \{\square \mid \rho_1, \ldots, \rho_m, \neg A_1, \ldots, \neg A_n\}$ implies $@(t_1, \ldots, t_k)$.

To complete the proof, we will show that Δ' can be converted into a derivation of a generalisation $\square \mid \gamma$ for the clause $\square \mid \rho_1, \ldots, \rho_m, \neg A_1, \ldots, \neg A_n$ from $DB' \cup KB \cup \{C \mid \neg @(X_1, \ldots, X_k)\}$. This is a corollary of a more general statement: if we can derive some clause D from clauses $C_1, \ldots, C_q$ in R, and $C'_1, \ldots, C'_q$ are some generalisations of those clauses, then there is a derivation from some of $C'_1, \ldots, C'_q$ in R of some generalisation D' of D. This can be easily proved by induction on the complexity of the derivation. Indeed, if the derivation contains some inferences, we apply the inductive hypothesis to the derivations of the premises of the last inference (resulting in D), deriving some generalisations of the premises. The induction step simply applies the generalisation-tolerance of R, possibly several times, to derive a generalisation of D from some of the new premises.

Finally, note that $\square \mid \gamma$ implies $\square \mid \rho_1, \ldots, \rho_m, \neg A_1, \ldots, \neg A_n$, and therefore $DB \cup \{\square \mid \gamma\}$ implies $@(t_1, \ldots, t_k)$.

4 Recording literals as search space pruning constraints.

Let us make an important observation: *some schematic answers to deductive queries cover no concrete answers*. These schematic answers are useless and the work spent on their generation is wasted. We can address this problem by trying to block search directions that can only lead to such useless schematic answers.

Suppose we are searching for schematic answers to $\langle DB, KB, \langle X_1, \ldots, X_k\rangle\langle Y_1, \ldots, Y_m\rangle C\rangle$ by deriving consequences of $DB' \cup KB \cup \{C \mid \neg @(X_1, \ldots, X_k)\}$ in an appropriate calculus, where DB' is an abstraction of DB.

Database abstraction literals. Suppose we have derived a clause $E = D \mid \rho'_1, \ldots, \rho'_p, \neg A_1, \ldots, \neg A_n$ where $p > 0, n \geq 0$, all the atoms A_i are of the form $@(t^i_1, \ldots, t^i_k)$ and all the literals ρ'_j are inherited from the recording literals of clauses from DB'. We can treat $\rho'_1, \ldots, \rho'_p$ as follows: if we can somehow establish that the constraint $\rho'_1, \ldots, \rho'_p$ *has no solutions* w. r. t. DB, we can remove the clause E from the search space. A *solution* of $\rho'_1, \ldots, \rho'_p$ w. r. t. DB is a substitution θ, such that all $\rho'_i\theta \in DB$.

Such a treatment can be justified with the following argument. It is obvious that if $\rho'_1, \ldots, \rho'_p$ has no solutions w. r. t. DB, then any more specific constraint $\rho'_1\sigma, \ldots, \rho'_p\sigma$, where σ is some substitution, also has no solutions. Since all recording literals are fully inherited in the calculi we are dealing with, any clause derived from E and any other clauses, will have the same property. Therefore, any schematic answer $\square \mid \gamma$ whose derivation contains the clause, will contain in γ a nonempty subconstraint without @, having no solutions w. r. t. DB. Thus, $\square \mid \gamma$ cannot cover any concrete answers because the non-@ part of the constraint γ cannot be satisfied.

To summarise, we can discard clauses like E without sacrificing the completeness w. r. t. concrete answers. Practically, this can be done by converting $\rho'_1, \ldots, \rho'_p$ into an SQL query (similar to how it is done in Section 5 for schematic answers) and evaluating the query on the database – empty result set indicates absence of solutions w. r. t. DB.

Answer literals. Suppose we have derived a schematic answer $\Box \mid D, \neg A_1, \ldots, \neg A_n$ where D only contains database abstraction literals or is empty, and $n > 0$. For the schematic answer to have instances, the answer literals $\neg A_i$ must be simultaneously unifiable. Indeed, suppose $@(t_1, \ldots, t_k)$ is an instance of the schematic answer. By Herbrand's theorem, $DB \cup \{\neg @(t_1, \ldots, t_k)\}$ is inconsistent with a finite set of ground clauses of the form $\Box \mid D\theta, \neg A_1\theta, \ldots, \neg A_n\theta$. We assume that the set is minimal. It cannot be empty because $@$ does not occur in DB and DB itself is trivially consistent. Consider any clause $\Box \mid D\theta, \neg A_1\theta, \ldots, \neg A_n\theta$ from the set. All the atoms $A_i\theta$ from this clause are equal to $@(t_1, \ldots, t_k)$ because otherwise the set would not be minimal – any model of the set without this clause could be extended to make this clause true by making an appropriate $A_i\theta$ true. Thus, all A_i are simultaneously unifiable.

The fact proved above can be used to prune the search space as follows: if we derive an intermediate clause with some $@$-literals that are not simultaneously unifiable, we can discard the clause because any schematic answer derived from it will have no instances. Moreover, we can use the most general unifier for $@$-literals to strengthen the test on database abstraction literals by applying the unifier to them before solving them on the database.

5 SQL generation.

Suppose that we have found a schematic answer $\Box \mid \rho_1, \ldots, \rho_p, \neg A_1, \ldots, \neg A_n$ to a query $\langle DB, KB, \langle X_1, \ldots, X_k\rangle\langle Y_1, \ldots, Y_m\rangle C\rangle$. Now our task is to enumerate all instances of the schematic answer by querying the relational database modeled by the fact set DB, with an SQL query.

We have four cases to consider. (1) If $p = n = 0$, then we simply have a refutation of KB. Formally, this means that any ground $@(t_1, \ldots, t_k)$ is a correct answer, but for practical purposes this is useless. Instead, we should simply inform the user about the inconsistency. (2) If $p = 0$ but $n \neq 0$, we have to try to unify all the literals A_i. If $\theta = mgu(A_1, \ldots, A_n)$, then the set of instances of the schematic answer coincides with the set of ground instances of $A_1\theta$. (3) If $p \neq 0$ but $n = 0$, there is a possibility that $DB \cup KB$ is inconsistent. We may want to check this possibility by checking if $\rho_1, \ldots, \rho_p$ has solutions over DB – if it does, DB is inconsistent with KB. The check itself can be done by converting $\rho_1, \ldots, \rho_p$ into an SQL query as in the next case, and checking if an answer to the SQL query exists. (4) In the rest of this section we will be considering the most interesting case when $p \neq 0$ and $n \neq 0$.

Using the considerations about answer literals from Section 4, we can prove that we only need to consider the case when $n = 1$. We can make another simplifying assumption: we only have to deal with schematic answers of the form $\Box \mid D, \neg @(X_1, \ldots, X_k)$, where X_i are pairwise distinct variables, each X_i occurs in D, and D contains only database abstraction literals. Enumeration of instances of more complex answer literals is reduced to this case.

Recall that all facts in DB are of the form $r_i(a_1^i, \ldots)$, where the predicates r_i correspond to tables in a relational database and all a_j^i are constants. Recalling Section 4, we can assume that literals from D do not contain compound terms.

Under these assumptions, it is straightforward to represent the schematic answer with a *semantically equivalent* clause of the form $E_a \vee E_c \vee E_d \vee D_x \vee A$, where (i) $A = @(X_1, \ldots, X_k)$ and all *answer variables* X_i are pairwise distinct; (ii) $D_x = \neg r_1(Y_1^1, \ldots, Y_{k(1)}^1) \vee \ldots \vee \neg r_p(Y_1^p, \ldots, Y_{k(p)}^p)$ and all variables Y_j^i are pairwise distinct; (iii) E_a consists of k negative equality literals $\alpha_i \not\simeq X_i$, $i = 1 \ldots k$, where $\alpha_i \in \{Y_1^1, \ldots, Y_{k(p)}^p\}$; (iv) E_c consists of zero or more negative equality literals of the form $\alpha \not\simeq \beta$, where $\alpha \in \{Y_1^1, \ldots, Y_{k(p)}^p\}$ and β is a constant; (v) E_d consists of zero or more negative equality literals of the form $\alpha \not\simeq \beta$, where $\alpha, \beta \in \{Y_1^1, \ldots, Y_{k(p)}^p\}$.

Finally, we transform the clause $E_a \vee E_c \vee E_d \vee D_x \vee A$ into an SQL query of the form $SELECT\ \langle columns \rangle\ FROM\ \langle tables \rangle\ WHERE\ \langle join\, conditions \rangle$, where $\langle columns \rangle$ maps X_i to table columns according to E_a, $\langle tables \rangle$ introduces aliases for all r_i (this is necessary because some of r_i may coincide), and $\langle join\ conditions \rangle$ is a conjunction of joins reflecting the conditions from E_c and E_d.

For a detailed algorithm for converting schematic answers to SQL, see [26])

6 A note on indexing Semantic Web documents with data abstractions.

In the context of Semantic Web (SW), it is important to be able to index distributed semantic data description sets (SW documents, for simplicity), so that, given a semantic query modulo some knowledge bases, we can load only the SW documents that are potentially relevant to the query. In this section we briefly sketch a possible scheme for such indexing that is compatible with our approach to deductive querying.

Conventional search engines index regular Web documents by words appearing in them. We cannot simply follow this example by indexing SW documents by the names of objects, concepts and relations occurring in them. This is so because retrieval in general may require reasoning, and thus the relevant documents may use no common symbols with the query. For example, a query may request to find animals of bright colours. If some SW document describes, e. g., pink elephants, it is relevant, but lexically there is no overlap with the query. Only reasoning reveals the relation between
"http://zooontology.org/concept#elephant" and
"http://zooontology.org/concept#animal", and between
"http://www.colors.org/concept#pink" and
"http://www.colors.org/concept#bright_colour".

Note that conceptually there is hardly any difference between RDBs and, say, OWL data description sets based on the Web: an RDB can be *modeled* as a set of ground atomic logical assertions, and, practically, an SW document *is* such a set. So, just like we use abstractions to represent relational data compactly in reasoning, we can use abstractions to represent SW documents. For example, a potentially large SW document introducing many pink elephants can be compactly represented by its abstraction $zoo{:}elephant(X) \mid zoo{:}elephant(X)$,

$colors{:}hasColour(X, Y) \mid colors{:}hasColour(X, Y)$ and
$colors{:}pink(X) \mid colors{:}pink(X)$.

It seems natural to use such abstraction clauses as indexes to the corresponding SW documents in a semantic search engine. Then, the query answering process can be organised as follows. As in the case of reasoning over RDB abstractions, a reasoner is used to derive schematic answers to a given query, based on all available abstractions of indexed SW documents. Each schematic answer to the query depends on some abstraction clauses. The documents associated with these clauses are potentially relevant to our query, so we download them, and only them, into our local RDB for further processing.

Of course, the indexing scheme presented here is just a conceptual one. The developers have the flexibility to chose a concrete representation – for example, they may just index by the URIs of concepts and relations, and only create the corresponding abstraction clauses when the reasoner is ready to inject them in the search space. There is also a possibility of adjusting the degree of generality of abstraction clauses by adding some ad hoc constraints. For example, the first of the abstraction clauses from the example above can be replaced with the more specific
$zoo{:}elephant(X) \mid zoo{:}elephant(X),\ pref(X,''http://www.elephants.com/'')$.
The ad hoc constraint $pref(X,''http://www.elephants.com/'')$ requires the prefix of the URI X to be "htpp://www.elephants.com/". The constraint is incompatible with, e. g., $pref(X,''http://www.rhinos.com/'')$, so if our reasoner derives a clause with these two constraints, it can safely discard it, thus improving the precision of indexing.

7 Related work.

We are not aware of any work that uses resolution-based reasoning in a way similar to the one proposed in this paper, i. e., for incremental query rewriting based on the use of complete query answering over database abstractions, implemented with constraints over the concrete data.

In general, semantic access to relational databases is not a new concept. Some of the work on this topic is limited to semantic access to, or semantic interpretation of relational data in terms of Description Logic-based ontologies or RDF (see, e. g., [10, 6, 4]), or non-logical semantic schemas (see [28]). There is also a large number of projects and publications on the use of RDB for storing and querying large RDF and OWL datasets: see, e. g., [24, 17, 11–13], to mention just a few. The format of the paper does not allow us to give a comprehensive overview of such work, so we will concentrate on research that tries to go beyond the expressivity of DL and, at the same time, is applicable to legacy relational databases.

The work presented here was originally inspired by the XSTONE project [31]. In XSTONE, a resolution-based theorem prover (a reimplementation of Gandalf, which is, in particular, optimised for taxonomic reasoning) is integrated with an RDBMS by loading rows from a database as ground facts into the reasoner and using them to answer queries with resolution. The system is highly scalable in terms of expressiveness: it accepts full FOL with some useful extensions, and also has parsers for RDF, RDFS and OWL. We believe that our approach has better data scalability and can cope with very large databases which are beyond the reach of XSTONE, mostly because our approach obtains answers in bulk, and also due to the way we use highly-optimised RDBMS.

Papers [23] and [22] describe, albeit rather superficially, a set of tools for mapping relational databases into OWL and semantic querying of the RDB. Importantly, the queries are formulated as SWRL [3] rule bases. Although SWRL only allows Horn rules built with OWL concepts, properties and equality, its expressivity is already sufficient for many applications. Given a semantic query in the form of a SWRL rule base, the software generates SQL queries in order to extract some relevant data in the form of OWL assertions and runs a rule engine on this data to generate final answers. So the reasoning is, at least partially, sensitive to the amount of data. This gives us hope that our approach can scale up better because the reasoning part of the process is completely independent of the concrete data.

Another project, OntoGrate [14], uses an approach to deductive query answering, which is based on the same ideas as ours: their FOL reasoner, OntoEngine [15], can be used to rewrite original queries formulated in terms of some ontology, into a finite set of conjunctive queries in terms of the DB schema, which is then converted to SQL. For this task, the reasoner uses *backward chaining with Generalised Modus Ponens* [29], which corresponds to negative hyperresolution on Horn clauses in the more common terminology. A somewhat ad hoc form of term rewriting [21] is used to deal with equality. Termination is implemented by setting some limits on chaining, which allows them to avoid incremental processing. We hope to go much further, mainly, but not only, by putting our work on a solid theoretical foundation. In particular, we are paying attention to completeness. Since our approach is based on well-studied calculi, we hope to exploit the large amount of previous research on completeness and termination, which seems very difficult to do with the approach taken by OntoEngine. Although we are very likely to make various concessions to pragmatics, we would like to do this in a controllable and reproducible manner.

On the more theoretical side, it is necessary to mention two other connections. The idea of using constraints to represent schematic answers is borrowed from Constraint Logic Programming [18] and Constrained Resolution [8]. Also, the general idea of using reasoning for preprocessing expressive queries into a database-related formalism, was borrowed from [20], where a resolution- and paramodulation-based calculus is used to translate expressive DL ontologies into Disjunctive Datalog. This work also shares a starting point with ours – the observation that reasoning methods that treat individuals/data values separately can not scale up sufficiently.

8 Future work.

Our future work will be mostly concentrated in the following directions:

Implementation and experiments. A proof-of-concept implementation has been already created, based on a version of the Vampire prover [27], and two experiments were done – one on a large instance of the LUBM benchmark [16] and another one on the BioCyc [19] dataset (in OWL). This exploratory work will be used to guide a more comprehensive implementation effort, including the implementation of a front-end for all first-order monotonic sublanguages of Derivation RuleML [2], an implementation of a client-server Java API and tuning the reasoner for the task of schematic answer derivation over RDB abstractions.

Equality treatment. If equality is present in our knowledge bases (e. g., in the form of OWL number restrictions), we can extend the standard superposition calculus to clauses with recording literals as we did with resolution. However, the completeness proof does not easily transfer to such use of superposition. Therefore, one of our main priorities now is to look for adjustments of the superposition calculus that would be provably complete w. r. t. schematic answers, without being too inefficient. An obvious obstacle to generalisation-tolerance is the absence of paramodulations into variables in the standard paramodulation-based calculi, so, for a start, we will try to use the specificity of reasoning over DB abstractions to eliminate the need for such inferences in generalisation-tolerant variants of superposition.

Completeness with redundancy deletion. Static completeness, proven in Section 3, is enough to guarantee that we will find all necessary answers only if our search procedure generates absolutely all possible derivations in the given calculus. In practice, such approach is almost always inefficient. Typically, some criteria are applied to detect redundant clauses and remove them from the current clause set (see, e. g., [5]).

It seems relatively easy to prove completeness of schematic answer derivation process in presence of the most important redundancy deletion technique: roughly, a clause subsumed by another clause can be deleted from the current clause set. The main idea for such a proof is that if subsumption removes an answer derivation from the search space, the search space will still contain a structurally simpler derivation of the same answer or a more general answer. Note that this is a property of generalisation-tolerant calculi. However, if we want to deal with equality efficiently, we have to demonstrate compatibility of our approach with the *standard redundancy criterion* (see, e. g., [5, 21]).

Termination. Very often it is desirable that a query answering implementation terminates on a given query having exhausted all solutions, e. g., for counting and aggregation of other kinds. We are interested in identifying combinations of practically relevant fragments of FOL with reasoning methods and strategies, that guarantee termination. For such fragments, complexity estimations may also be useful.

Acknowledgements. Many thanks to RuleML for supporting the publication, and to Harold Boley for numerous productive discussions on the subject, especially for drawing my attention to modern rule languages.

References

1. Rule Interchange Format WG Charter. http://www.w3.org/2005/rules/wg/charter.html.
2. The Rule Markup Initiative Web Site. http://www.ruleml.org/.
3. W3C SWRL Submission:. http://www.w3.org/Submission/SWRL/.
4. A.Ranganathan and Z. Liu. Information Retrieval from Relational Databases using Semantic Queries. In *Proc. ACM CIKM*, pages 820–821, 2006.
5. L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In *Hand. of Automated Reasoning, vol. I*. 2001.
6. C. Bizer. D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs. In *ISWC 2004*.

7. Harold Boley. The RuleML Family of Web Rule Languages. In *PPSWR06*, 2006.
8. H.-J. Bürckert and W. Nutt. On Abduction and Answer Generation through Constrained Resolution. Technical Report DFKI RR-92-51, 1992.
9. D. T. Burhans and S. C. Shapiro. Defining Answer Classes Using Resolution Refutation. *Journal of Applied Logic*, 5:70–91, 2007.
10. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati. MASTRO-I: Efficient Integration of Relational Data through DL Ontologies. In *DL-07*, 2007.
11. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable Description Logics for Ontologies. In *AAAI'05*, pages 602–607, 2005.
12. C. M. Chen, V. Haarslev, and J. Y. Wang. LAS: Extending Racer by a Large Abox Store. In *DL-2005*.
13. J. Dolby, A. Fokoue, A. Kalyanpur, L. Ma, C. Patel, E. Schonberg, K. Srinivas, and X. Sun. Efficient reasoning on large SHIN Aboxes in relational databases. 2007. (to appear).
14. 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 Database at ICDE 2006*, 2006.
15. D. Dou, D. McDermott, and P. Qi. Ontology Translation on the Semantic Web. *Journal of Data Semantics*, 2:35–37, 2005.
16. Y. Guo, J. Heflin, and Z. Pan. An Evaluation of Knowledge Base Systems for Large OWL Datasets. In *ISWC 2004*, pages 613–627, 2004.
17. I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The Instance Store: Description Logic Reasoning with Large Numbers of Individuals. In *DL'04*, 2004.
18. J. Jaffar and M. J. Maher. Constraint Logic Programming: a Survey. *Journal of Logic Programming*, 19(20):503–581, 1994.
19. Markus Krummenacker, Suzanne Paley, Lukas Mueller, Thomas Yan, and Peter D. Karp. Querying and Computing with BioCyc Databases. *Bioinformatics*, 21(16):3454–3455, 2005.
20. B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD Thesis, 2006.
21. R. Nieuwenhuis and A. Rubio. Paramodulation-Based Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I. 2001.
22. M.J. O'Connor, R.D. Shankar, S.W. Tu, C. Nyulas, A.K. Das, and M.A. Musen. Efficiently Querying Relational Databases Using OWL and SWRL. In *RR 2007*, 2007.
23. M.J. O'Connor, R.D. Shankar, S.W. Tu, C. Nyulas, D.B. Parrish, M.A. Musen, and A.K. Das. Using Semantic Web Technologies for Knowledge-Driven Querying of Biomedical Data. In *AIME 07*, 2007.
24. Z. Pan and J. Heflin. DLDB: Extending Relational Databases to Support Semantic Web Queries. In *Workshop on Practical and Scaleable Semantic Web Systems, ISWC 2003*, 2003.
25. R. Ramakrishnan and J. Gehrke. *Database Management Systems, third edition*. McGraw-Hill, 2003.
26. A. Riazanov. Resolution-based Query Answering for Semantic Access to Relational Databases: A Research Note. Preprint 0901.0339, ArXiv, January 2009.
27. A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.
28. N. Rishe. SemanticSQL: A Semantic Wrapper for Relational Databases. http://n1.cs.fiu.edu/semantic.wrapper.pdf, 2004. (white paper).
29. S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach, Second Edition*. Prentice-Hall, Inc., 2003.
30. T. Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199–204, 1997.
31. T. Tammet, V. Kadarpik, H.-M. Haav, and M. Kaaramees. A Rule-based Approach to Web-based (Database) Application Development. In *7th International Baltic Conference on Databases and Information Systems*, pages 202–208, 2006.