SQOWL2: Transactional Type Inference for OWL 2 DL in an RDBMS

Yu Liu and Peter McBrien

Department of Computing, Imperial College London
180 Queens Gate, London, UK
{yu.liu11,p.mcbrien}@imperial.ac.uk
http://www.doc.ic.ac.uk

Abstract. SQOWL2 is a compiler which allows an RDBMS to support sound reasoning of $\mathcal{SROIQ}(\mathcal{D})$ **description logics**, by implementing ontologies expressed in the OWL 2 DL language as a combination of tables and triggers in the RDBMS. The reasoning process is divided into two phases of **classification** of the **T-Box** and **type inference** of the **A-Box**. SQOWL2 establishes a relational schema based on classification completed using the Pellet reasoner, and performs type inference by using **SQL triggers**. SQOWL2 supports type inference over all OWL 2 DL constructs, and supports a more conventional relational schemas, rather than naively mapping OWL classes and properties to relational tables with one and two columns. Moreover, SQOWL2 is a transactional reasoning system (with full ACID properties), since the results of reasoning are available within the same transaction as that in which the base data of the reasoning was inserted.

Keywords: OWL 2 DL, Description Logics, Ontologies, Reasoning, Type Inference, Database Triggers, ACID Transactions

1 Introduction

The most common storage method used in mainstream data processing is to use a **relational database management system** (**RDBMS**), which have a number of well known and understood properties. One key property is the ACID property of transactions [6], which in outline, states that a group of operations grouped together into a transaction should occur as an atomic action, and for which once the results are reported, they are durable after a system crash. In the context of reasoning on ontologies, the notion of transactions may be naturally extended to the notion of **transactional reasoning** [12], where the results of reasoning derived from data changed by the database operations should be available as part of the atomic action of the transaction.

To illustrate the concept of transactional reasoning, consider the description logic rule $\mathsf{Father} \equiv \mathsf{Man} \sqcap \mathsf{Parent}$. If the data about these classes were held in a RDBMS, and John was recorded as a Man, then if a transaction added that John was a Parent, then any query on the result of the transaction should also be able

to view that John is a Father. However, most approaches to storing ontology data in an RDBMS will make the process of reasoning be detached from transaction processing of the data. Specifically, in our example, after the transaction that added John was a Parent, the database could be queried and find that he was not a Father until a separate process of reasoning had derived that fact. Hence we would say that these approaches do not support transactional reasoning.

Our work focuses on reasoners that support the OWL language, and in particular the OWL 2 DL version [13]. To achieve transactional reasoning, two approaches were identified in [12]: **view based reasoning** (**VBR**), where rules are used to derive the result of reasoning as each query is executed over the database, and **trigger based reasoning** (**TBR**) where triggers (active rules) are used to materialise the result of reasoning at data insert time, and queries simply read the materialised views. As discussed in [12], the advantages and disadvantages of using views or materialised views are well known, and each serve different real world requirements. In the context of reasoning, one specific advantage of the TBR approach is that reasoning is incremental, and supports very fast query processing. Examples of VBR approaches include DLDB2 [14] and DBOWL [2], and of TBR include SQOWL [11]. This paper describes a substantial improvement on SQOWL, called SQOWL2.

To illustrate the issues addressed in this paper, we introduce the **family** ontology[1]. It uses almost all the new OWL 2 features [7]. The family ontology includes the following OWL 2 DL rules of T-Box (which contain classes that by convention begin with upper case letters, and properties which by convention start with lower case letters):

$$\mathsf{Father} \equiv \mathsf{Man} \sqcap \mathsf{Parent} \quad (1)$$
$$\mathsf{Mother} \equiv \mathsf{Woman} \sqcap \mathsf{Parent} \quad (2)$$
$$\mathsf{Parent} \equiv \mathsf{Father} \sqcup \mathsf{Mother} \quad (3)$$
$$\mathsf{Father} \sqcap \mathsf{Mother} \sqsubseteq \bot \quad (4)$$
$$\top \sqsubseteq \forall \mathsf{hasHusband}.\mathsf{Man} \quad (5)$$
$$\top \sqsubseteq \forall \mathsf{hasHusband}^{-}.\mathsf{Woman} \quad (6)$$
$$\top \sqsubseteq \forall \mathsf{hasSpouse}.\mathsf{Person} \quad (7)$$
$$\top \sqsubseteq \forall \mathsf{hasSpouse}^{-}.\mathsf{Person} \quad (8)$$
$$\top \sqsubseteq \forall \mathsf{hasRelative}.\mathsf{Person} \quad (9)$$
$$\top \sqsubseteq \forall \mathsf{hasRelative}^{-}.\mathsf{Person} \quad (10)$$
$$\top \sqsubseteq \forall \mathsf{loves}.\mathsf{Person} \quad (11)$$
$$\top \sqsubseteq \forall \mathsf{loves}^{-}.\mathsf{Person} \quad (12)$$
$$\top \sqsubseteq \forall \mathsf{hasParent}.\mathsf{Person} \quad (13)$$
$$\top \sqsubseteq \forall \mathsf{hasParent}^{-}.\mathsf{Person} \quad (14)$$
$$\top \sqsubseteq \forall \mathsf{hasGrandParent}.\mathsf{Person} \quad (15)$$
$$\top \sqsubseteq \forall \mathsf{hasGrandParent}^{-}.\mathsf{Person} \quad (16)$$
$$\top \sqsubseteq \exists \mathsf{hasRelative}.\mathsf{Self} \quad (17)$$
$$\top \sqsubseteq \neg\exists \mathsf{hasParent}.\mathsf{Self} \quad (18)$$
$$\mathsf{hasParent} \sqcap \mathsf{hasParent}^{-} \sqsubseteq \bot \quad (19)$$
$$\mathsf{hasSpouse}^{-} \sqsubseteq \mathsf{hasSpouse} \quad (20)$$
$$\mathsf{hasHusband} \sqsubseteq \mathsf{hasSpouse} \quad (21)$$
$$\mathsf{hasParent} \sqcap \mathsf{hasSpouse} \sqsubseteq \bot \quad (22)$$
$$\mathsf{Narcissist} \equiv \exists \mathsf{loves}.\mathsf{Self} \quad (23)$$
$$\mathsf{hasParent} \circ \mathsf{hasParent} \sqsubseteq \mathsf{hasGrandParent} \quad (24)$$

For example, rule (1) states that fathers are the intersection of things that are both a man and a parent, and rule (2) states mothers are the intersection of things that are both a woman and a parent. Rule (6) shows that the domain for the hasHusband property is a Woman, and in (5) its range is a Man. Moreover, we list several examples of OWL 2 DL constructors. For instance, reflexive, irreflexive, asymmetric and symmetric properties can be demonstrated by rule (17),

[1] http://www.doc.ic.ac.uk/∼yl12510/family.owl

(18), (19) and (20), respectively. Subsumption, disjointness, self restriction and property chains can be described by rule (21), (22), (23) and (24), respectively.

In addition to the T-Box, data in an OWL 2 DL ontology is called the A-Box, and is stated as grounded predicates over the classes or properties of the T-Box. Examples of such data in the family ontology are:

$$\mathsf{Father}(\mathsf{John}) \quad (25) \qquad \mathsf{hasHusband}(\mathsf{Mary}, \mathsf{John}) \quad (27)$$
$$\mathsf{Mother}(\mathsf{Mary}) \quad (26) \qquad \mathsf{hasParent}(\mathsf{Lewis}, \mathsf{Albert}) \quad (28)$$
$$\mathsf{hasParent}(\mathsf{Albert}, \mathsf{Alex}) \quad (29)$$

In the SQOWL approach presented in [11], a very simple process was used where the OWL classes and properties are mapped to unary and binary relations, as shown in Example 1.

Example 1.

Father(id)	Man(id)
Parent(id)	Mother(id)
Woman(id)	Person(id)
Narcissist(id)	
hasHusband(domain,range)	hasGrandParent(domain,range)
hasSpouse(domain,range)	hasRelative(domain,range)
hasParent(domain,range)	loves(domain, range)

The SQOWL approach then allowed the use of any OWL reasoner such as FacT++ [3] or Pellet [15] to perform **classification** of the T-Box, which was then used to build trigger rules over the relational schema. The **type inference** over the A-Box then occurs as data is inserted into the RDBMS by the execution of the triggers. For example, in classification, from rules (1) and (2) six rules can be inferred:

$$\mathsf{Father} \sqsubseteq \mathsf{Man} \quad (30) \qquad \mathsf{Mother} \sqsubseteq \mathsf{Woman} \quad (33)$$
$$\mathsf{Father} \sqsubseteq \mathsf{Parent} \quad (31) \qquad \mathsf{Mother} \sqsubseteq \mathsf{Parent} \quad (34)$$
$$\mathsf{Man} \sqcap \mathsf{Parent} \sqsubseteq \mathsf{Father} \quad (32) \qquad \mathsf{Woman} \sqcap \mathsf{Parent} \sqsubseteq \mathsf{Mother} \quad (35)$$

From this classified schema, triggers were generated. For example, taking the rules (30)–(32) above, we can derive four logical triggers (which we will detail the syntax of in Section 3.2) of the form:

when $^{+}\mathsf{Father}(x)$ **then** $\mathsf{Man}(x)$, **when** $^{+}\mathsf{Parent}(x)$ **if** $\mathsf{Man}(x)$ **then** $\mathsf{Father}(x)$
when $^{+}\mathsf{Father}(x)$ **then** $\mathsf{Parent}(x)$, **when** $^{+}\mathsf{Man}(x)$ **if** $\mathsf{Parent}(x)$ **then** $\mathsf{Father}(x)$

The SQOWL2 approach presented in this paper provides the following significant improvements over the original SQOWL approach in [11].

1. SQOWL2 supports sound reasoning of the more recent OWL 2 DL standard, as opposed to OWL 1 DL support by SQOWL. In particular SQOWL2 supports keys, property chains, and reflexive properties (Type inference of qualified cardinality restrictions is introduced in [11]).
2. SQOWL2 gathers together all the reasoning rules associated with a particular concept into a single trigger statement, rather than producing a separate trigger statement per rule. Not only is this more efficient, but also in allows SQOWL2 to generate triggers for those DBMS such as Microsoft's SQL Server that only support one trigger (of a given type) per table.

3. SQOWL2 uses the fact (originally from [1]) that a property is functional to allow the property to be stored in the same table used to stored class that is the domain of the property. This results in SQL schemas that are like those one would produce via a conventional design process into 3NF schema, and will allow SQOWL2 to be adapted so that it can be used to generate triggers on top of existing relational schemas (as opposed to generating new schemas with associated triggers).

The remainder of the paper gives more detail of the operation and performance of SQOWL2. Section 2 gives an overall architecture of SQOWL2, which provides a complete system for building databases with triggers from specifications in OWL 2 DL. Section 3 describes the details of the process used in generating SQL triggers to implement the OWL 2 DL type inference process. Section 4 then evaluates SQOWL2 on how the SQOWL2 tool can be used, and its performance over some benchmark datasets. Finally, Section 5 compares SQOWL2 with several related systems, and is followed by a conclusion section.

2 SQOWL2 Architecture

SQOWL2 uses this division of the OWL 2 DL reasoning process (**classification** of T-Box and **type inference** of A-Box) to build a reasoning system in several steps, the overall design for which is illustrated in Figure 1.

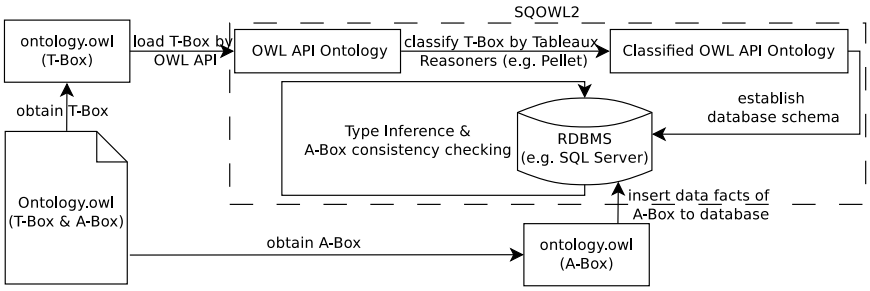


Fig. 1. The architecture of SQOWL2.

The T-Box (schema) from the ontology is then read using the OWL API [8]. Any A-Box (data) is also read and held until after the relational schema and triggers have been created. In the current prototype, classification of the T-Box (i.e. reasoning over the schema) is performed using Pellet, though in principle any other reasoner could be used. The revised T-Box is then used to (a) produce a relational schema, and (b) define triggers over that schema, which will perform the reasoning outlined in the previous section.

Once the schema and triggers have been created, the relational database is now capable of performing type inference over the A-Box (i.e. reasoning over

the data). If the OWL 2 DL ontology contained an A-Box, then this is inserted into database as the last phase of the SQOWL2 execution. Then the relational database becomes a free standing type inference system, when inserts made using SQL resulting in a type inference being performed, and derived data being inserted into the database.

3 From OWL 2 DL to Triggers

We now outline in more detail the reasoning process followed by SQOWL2 in first generating a database schema, then deriving logical triggers over that schema. For reasons of space we do not detail the mapping of the triggers in SQL statements, but this process is very straightforward.

3.1 Building a traditional RDB schema

SQOWL2 builds the relational database schema in two stages, which depends on the classification results from Pellet. The first step builds a canonical relational schema (Example 1), where each class maps to a unary relation, and each property to a binary relation (i.e. a class is mapped to a relational table with one column, and a property is mapped to a relational table with two columns). Moreover, subclass relationships should be mapped as foreign keys.

The second stage is to map the canonical schema to a physical relational schema, by taking account of **functional properties** (cardinality restriction is no more than one) and **key properties** (properties that together can uniquely identify an object) and in the OWL 2 DL ontology. In general, if a class $C(x)$ has n functional properties (or key properties): $P_1(x, y_1), \ldots, P_n(x, y_n)$ that contain C as their domain, then we store the class and properties as a single table $C(x, y_1, \ldots\ldots, y_n)$.

For example, if hasHusband and hasSpouse are denoted as functional properties in the family ontology, and hence are not stored as separate tables, but instead are stored in the table representing the domain, giving the following modified schema as Example 2.

Example 2.

Father(id)	Man(id)
Parent(id)	Mother(id)
Narcissist(id)	
Woman(id,hasHusband)	Person(id,hasSpouse)
hasParent(domain,range)	hasGrandParent(domain,range)
hasRelative(domain,range)	loves(domain,range)
Father(id)$\stackrel{fk}{\Rightarrow}$Man(id)	Father(id)$\stackrel{fk}{\Rightarrow}$Parent(id)
Mother(id)$\stackrel{fk}{\Rightarrow}$Woman(id)	Mother(id)$\stackrel{fk}{\Rightarrow}$Parent(id)

3.2 Type inference by generating SQL triggers

Type inference needs to occur every time a data value is inserted into the database schema, which is achieved by having SQL triggers invoke queries and

updates after each insert. For example, rule (30) implies all instances in Father should also appear in Man. In this case, if data John is inserted into table Father, a trigger needs to exist to insert John to table Man.

SQL triggers are translated from constructors of OWL 2 DL ontologies according to logical triggers. The logical triggers are of the general form:

$$\textit{OWL 2 DL construct}: \rightsquigarrow \textbf{when } \textit{event} \textbf{ if } \textit{condition} \textbf{ then } \textit{action}.$$

where *event* identifies the insertion of a particular class or property in the database. There are two types of event: if *event* is prefixed with $^-$ then *condition* and *action* are executed before the insertion, whilst if *event* is prefixed with $^+$ then *condition* and *action* are executed after the insertion.

SQL **before triggers** (PL/pgSQL)/**instead of triggers** (Transact SQL) are used to implement $^-$ events, and **after triggers**, used for $^+$ events. The *condition* is a Datalog query over the database, and *action* is one of *insert*, *ignore* and *reject*.

One basic rule deals with the notion that because of the open world nature of reasoning, we might repeatedly infer the same fact, and thus we have to prevent duplicate updates to a table. This is implemented by the logical trigger:

class: C $\rightsquigarrow$ **when** $^-C(x)$ **if** $C(x)$ **then** *ignore*

This means that when a value is inserted into a table, before the actual insert is done, a check is made to determine if the value is already present in the table, and if so, the insert is ignored.

Another simple logical trigger deals with subclass rules such as (30) and (31):

subClassOf: $\mathsf{C} \sqsubseteq \mathsf{D} \rightsquigarrow$ **when** $^+C(x)$ **if** true **then** $D(x)$

which ensures that any insert to a subclass is inferred to also be an insert to the superclass. Similarly, the logical trigger for subproperty rule such as (21) is:

subPropertyOf: $\mathsf{P} \sqsubseteq \mathsf{Q} \rightsquigarrow$ **when** $^+P(x,y)$ **if** true **then** $Q(x,y)$

A new feature of OWL 2 DL (compared to OWL 1 DL) is the **property chain**, which allows for a property to be defined from the concatenation of two or more other properties. For example, one can define that a parent of a parent is a grandparent by rule (24). The logical trigger for a property chain (contains two subchain properties) is:

propertyChain: $\mathsf{P_1} \circ \mathsf{P_2} \sqsubseteq \mathsf{P}$

$$\rightsquigarrow \textbf{when } {}^+P_1(x,y) \textbf{ if } P_2(y,z) \textbf{ then } P(x,z)$$
$$\rightsquigarrow \textbf{when } {}^+P_2(y,z) \textbf{ if } P_1(x,y) \textbf{ then } P(x,z)$$

Hence in the first rule, an insert to the first property in the chain causes a check for matched data in the second property in the chain, and if found causes an insert to the chained property table. We also generalise the above logical trigger to handle the situation that a chain property contains n subchain properties:

propertyChain: $\mathsf{P_1} \circ \ldots \circ \mathsf{P_i} \circ \ldots \circ \mathsf{P_n} \sqsubseteq \mathsf{P}$

$$\rightsquigarrow \textbf{when } {}^+P_1(x,y) \textbf{ if } P'_{2,n}(y,z) \textbf{ then } P(x,z)$$
$$\rightsquigarrow \textbf{when } {}^+P_n(y,z) \textbf{ if } P'_{1,n-1}(x,y) \textbf{ then } P(x,z)$$
$$\rightsquigarrow \textbf{when } {}^+P_i(p,q) \textbf{ if } P'_{1,i-1}(x,p), P'_{i+1,n}(q,z) \textbf{ then } P(x,z)\ _{1<i<n}$$

where $P'_{m,n}(x,y) =$

$$\pi_{P_m.domain,P_n.range}\ \sigma_{P_j.range=P_{j+1}.domain}(P_m \times \ldots \times P_n)\ _{m\leq j<n}$$

Therefore, the first and second situations mean that if there is an insertion to the first or last subchain, the trigger will treat the remaining subchains as a join unit and search data matched inside the unit. The third scenario handles the insertion to the middle subchains (i.e. not the first or the last subchain) by creating two join units and then fetching matching data from them.

For example, in family ontology, rule (36) defines one subset of hasBrotherInLaw that:

$$\mathsf{hasSibling} \circ \mathsf{hasSpouse} \circ \mathsf{hasBrother} \sqsubseteq \mathsf{hasBrotherInLaw} \tag{36}$$

Thus, if hasSibling(Kate,Mary), hasBrother(John,Lewis), and an insertion (Mary, John) to table Man, the logic trigger will infer a new insertion (Kate, Lewis) to table hasBrotherInLaw.

Also introduced in OWL 2 DL is the ability to define a **reflexive property**, an **irreflexive property** and a **self restriction** by allowing a special class Self to be used in the range of a property (that corresponds to class in the domain). The domain of a reflexive property is always the whole universe. For example, rule (17) states property hasRelative is reflexive in which anyone is a relative of them-self. The negative side of reflexivity is irreflexivity, which means no individual is related to itself by a irreflexive property, such as hasParent defined in rule (18). The logical triggers for reflexive & irreflexive properties are:

reflexiveProperty: $\top \sqsubseteq \exists \mathsf{P}.\mathsf{Self} \rightsquigarrow$ **when** ${}^{+}\top(x)$ **if** $\neg P(x,x)$ **then** $P(x,x)$

irreflexiveProperty: $\top \sqsubseteq \neg\exists \mathsf{P}.\mathsf{Self} \rightsquigarrow$ **when** ${}^{-}P(x,x)$ **if** true **then** *reject*

In the case of reflexive properties, since their domain is always the whole universe, each insert x in an ontology entails that (x,x) also needs to be inserted to this property table. In contrast, in irreflexive properties, tuples like (x,x) are not allowed to be inserted and should be rejected.

Compared with reflexive properties, self restriction means the domain is not always the whole universe. For example, rule (23) defines a new class Narcissist in which all individuals love (e.g. property loves) themselves (i.e. the reflexivity is limited only to Narcissist). The logical trigger for self restriction is:

selfRestriction: $\mathsf{C} \equiv \exists \mathsf{P}.\mathsf{Self}$

$\rightsquigarrow$ **when** ${}^{+}P(x,y)$ **if** $x = y, \neg C(x)$ **then** $C(x)$

$\rightsquigarrow$ **when** ${}^{+}C(x)$ **if** $\neg P(x,x)$ **then** $P(x,x)$

This trigger enables that if a property tuple (x,x) is inserted to the self restricted property, the data x should also be inserted to the related class. Moreover, if there is an data x in the related class, the property tuple (x,x) should be inferred and inserted to the property.

In addition to the above new features, OWL 2 DL also improves its ability of expressiveness by adding other constructors, such as disjoint union (rules (3) and (4)) between classes, asymmetric (rule (19)) and negative relations among properties. The logical triggers for a disjoint union and an asymmetric property are listed as follows.

disjointUnion: $\mathsf{CN} \equiv \mathsf{C}_1 \sqcup \ldots \sqcup \mathsf{C}_n, \mathsf{C}_i \sqcap \mathsf{C}_j \sqsubseteq \bot_{1 \le i,j \le n, i \ne j}$

$\rightsquigarrow$ **when** ${}^{-}C_i(x)_{1 \le i \le n}$ **if** $foreach(1 \le j \le n, j \ne i), \neg C_j(x)$ **then** $C_i(x)$

if $\neg CN(x)$ **then** $CN(x)$.

asymmetricProperty: $\mathsf{P} \sqcap \mathsf{P}^{-} \sqsubseteq \bot \rightsquigarrow$ **when** ${}^{-}P(x,y)$ **if** $P(y,x)$ **then** *reject*

The trigger for disjointUnion only allows insertion if the data which will be inserted to a class does not appear in its disjoint classes (the trigger for disjoint properties is in a similar way with properties). Moreover, in disjointUnion trigger, any inserted data which does not exist in the union class will be inserted to the union class. Furthermore, the asymmetricProperty trigger does not allow tuple (x, y) existing with its asymmetric tuple (y, x).

Note that if $C(x, y_1, \ldots, y_i, \ldots, y_n)$ is used in the physical relational schema to represent functional property $P(x, y_i)$ from the canonical relational schema, then all references to $P(x, y_i)$ in the rules are replaced by $C(x, y_1, \ldots, y_i, \ldots, y_n)$. For instance, if as illustrated in the previous subsection, properties hasHusband and hasSpouse are added as additional columns to table Woman and Person, respectively, then references to hasHusband and hasSpouse are changed to Woman and Person.

4 Evaluation of SQOWL2

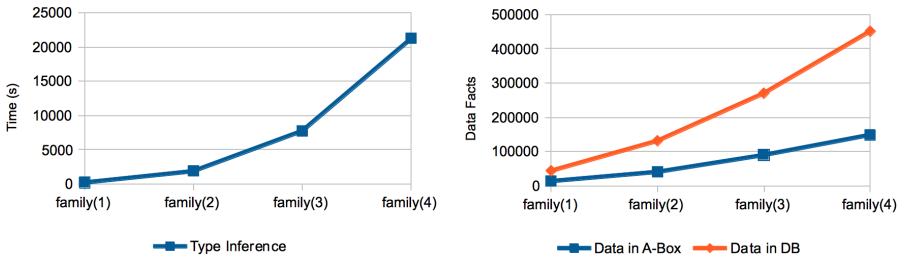
The SQOWL2 system can be evaluated in three steps. Firstly, in order to show that the system is able to perform type inference over a large A-Box, three benchmark ontologies are loaded into SQOWL2, and the total load and reasoning times are reported. Secondly, in order to evaluate the reasoning completeness, we compare the reasoning performance between SQOWL2 and two Tableaux reasoners. Finally, we evaluate the performance of incremental consistency checking.

4.1 Execution times of SQOWL2 over Large A-Boxes

Three example ontologies have been used to test and benchmark SQOWL2. The first ontology has been produced by us, called the **family** ontology, improved from [7] and formed by amalgamating fragments from the OWL 2 standard, together with an A-Box generator. The family ontology benchmark used in the evaluation provides four different ontologies with increasing size of A-Box: family(1) contains an A-Box of 13,871 facts (i.e. class and property instances), family(2) 40,854 facts, family(3) 90,104 facts, and family(4) 148,867 facts.

The other two ontologies are the well known **Lehigh University Ontology Benchmark** (**LUBM**) [4] and **University Ontology Benchmark** (**UOBM**) [10]. To be more specific, LUBM(1), which has about 100,000 facts, is used in this evaluation. For the UOBM, the OWL-DL version is applied.

SQOWL2 was used with Pellet for T-Box reasoning to produce tables and triggers on a Microsoft SQL Server 2005 database, on a machine with 8 cores, 2 Intel Xeon E5345 2.33GHz CPUs, and 8GB of memory. The performance results of type inference on family ontologies are shown in Figure 2. From the graphs, it can be seen that SQOWL2 is able to perform transactional reasoning at an average rate of 100 fact inserts per second for a 150,000 fact ontology, in the process inferring an additional 300,000 facts. Since the ontology is very small, the time taken for creating the database tables and triggers was only 0.6 seconds.



(a) Type Inference Time (b) Data Inference Amount

Fig. 2. Timing of SQOWL2 on family ontologies.

SQOWL2 were also tested against the LUBM and UOBM benchmarks. These two benchmarks contain only OWL 1 constructors; therefore, the benchmarks are essentially testing less complex reasoning. The performance of reasoning is shown in Table 1, from which is can be seen that SQOWL2 is able to process over 150 inserts per second on average.

Table 1. Reasoning performance of SQOWL2 on LUBM & UOBM.

Action	LUBM	UOBM
Loading T-Box	1.43s	1.42s
Classification	0.41s	0.64s
Building DB schema	0.38s	0.77s
Type inference	255.68s	1250.08s
Data facts in A-Box	100,543	260,580
Facts in DB after reasoning	137,920	279,436

4.2 Comparison with Tableaux Reasoners

HermiT 1.3.6 and Pellet 2 are two Tableaux reasoners, which support reasoning over OWL 2 DL. We expect Tableaux reasoners to provide more complete results at the expense of execution time. To evaluate the reasoning efficiency and completeness of SQOWL2, we compare the performance results between SQOWL2 and Tableaux reasoners in two ways: time spent on reasoning and the amount of data after reasoning. The **Wine** ontology [2] is a more complicated OWL 1 DL ontology, but does not come with any data generator (just around 500 A-Box facts). In addition, a family ontology (family(0)) which contains a small A-Box (1491 A-Box facts) is also used to show the performance of SQOWL2. The comparison of reasoning (which includes classification and type inference)between SQOWL2 and Hermit & Pellet is illustrated in Figure 3. As expected, SQOWL2

[2] www.w3.org/TR/owl-guide/wine.rdf

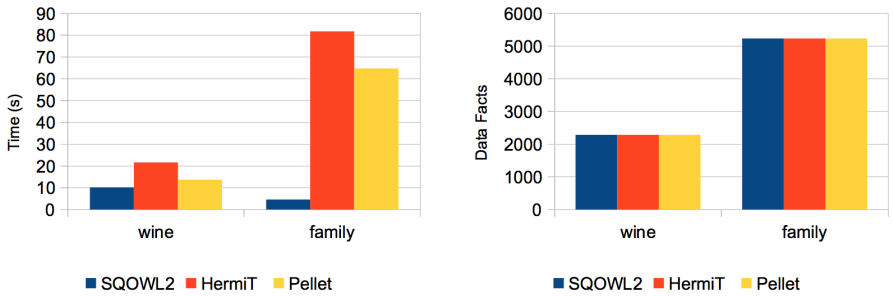


Fig. 3. Comparison of reasoning performance.

takes less time, but shows a very high reasoning completeness level. For wine ontology, SQOWL2 only spent 10.2s on reasoning (Hermit uses 21.6s), but generates 2277 A-Box instances which is 100% of the results found by Hermit. For family ontology, the reasoning is more efficient and the completeness level is also 100%.

4.3 Incremental Consistency Checking of Adding Information

Incremental consistency checking happens when the A-Box is updated. In this evaluation of incremental reasoning, we only focus on adding new individuals and property tuples rather than the deleting or other update operations. To illustrate, we could have a new A-Box fact hasWife(Lewis,Susan) added to the family ontology. In Pellet, this fact can be added directly by a Pellet API, and the incremental reasoning time of Pellet should be recorded. However, in SQOWL2, ontologies are translated to database schema. Therefore, the new axiom can be translated to a SQL insertion that,

```
INSERT INTO Man VALUES ('Lewis','Susan')
```

By comparing the incremental consistency checking time of adding this axiom using Pellet and the time to execute the above SQL statement. SQOWL2 shows a very faster speed of incremental reasoning of adding information. Execution times of such an insert on a fully populated version of the family(1) database (i.e. one with 5,033 A-Box instances) are about 0.005s on the test system. By contrast, Pellet needs about 9 seconds.

5 Related Work

There are different kinds of DL reasoners. Tableaux reasoners which are using tableaux algorithms, such as Pellet, Racer [5] and FacT++, are the most common used, since they are efficient on classification of a T-Box and consistency checking of an A-Box. However, they are not efficient at handling ontologies with a large A-Box.

Compared with Tableaux reasoners, there is an alternative way of performing reasoning by using reasoning rules (rule based reasoning), in order to handle large scale of ontologies. According to the description from the Introduction part, rule based reasoning systems may be divided into VBR, TBR and **application based reasoner** (**ABR**) [12] depending on the methods they used to implement the rules. VBR systems, such as DLDB2 and DBOWL use SQL views to achieve type inference, while SQOWL2, a TBR system applies SQL triggers to infer new information based on OWL constructors. ABR systems, such as SOR [9, 16], uses reasoners to implement type inference outside a DB.

SOR (called Minerva previously), as an example of ABR, uses several tableaux based reasoners to classify the T-Box first and then apply SQL statements to apply description logic program rules, in order to implement its type inference engine. It materialises reasoned ontologies inside an RDBMS, which shows a faster querying process. However, current version of SOR only supports OWL 1 DL. Since it only maps a reasoned ontology into a relational schema. It does not allow type inference inside the database; therefore, SOR does not support transactional reasoning and incremental reasoning inside an RDBMS.

Moreover, since in DBOWL, views are non-materialised, it cannot allow queries running in the speed as SQOWL2, which materialised the inferred information. In addition, VBR systems known to us do not support type inference on OWL 2 DL features. For instance, DBOWL and DLDB2 both only cover OWL 1 DL features. Furthermore, SQOWL2 establishes a more traditional relational schema which is a 3NF schema, by translating functional properties and key properties as additional attributes to the properties' domain tables.

6 Conclusions

SQOWL2 is the first transactional reasoning system that supports sound reasoning of OWL 2 DL, and which supports processing of reasoning over standard relational schemas, where functional properties of a class are all held together in a single relation. The evaluation shows that SQOWL2 is substantially faster than Pellet and HermiT, yet provides result sets which are as complete for the benchmark ontologies. It is also shown to be much faster at incremental consistency checking, and since it materialises the reasoned data, is fast as query processing. Of course, the approach is unsuited to applications were the inferred data is very large, and queries are relatively infrequent compared to updates.

Whilst other reasoners exist that are optimised for performance over large A-Boxes, none of these support transactional reasoning, nor are effective at incremental reasoning. Naturally, the advantages of SQOWL2 have to be balanced with its disadvantages. Firstly, since the results of reasoning are being materialised as data is inserted, certain applications might generate very large materialised views from relatively small original datasets. Secondly, our work has not yet addressed the key issue of handling deletes from the database effectively, which is the subject of our current work.

References

1. A. Borgida and R. J. Brachman. Loading data into description reasoners. In *ACM SIGMOD Record*, volume 22, pages 217–226. ACM, 1993.
2. M. del Mar Roldan-Garcia and J. F. Aldana-Montes. Evaluating DBOWL: A Non-materializing OWL Reasoner based on Relational Database Technology. In *OWL Reasoner Evaluation Workshop (ORE 2012)*.
3. FacT++. http://owl.man.ac.uk/factplusplus/.
4. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2–3):158–182, 2005.
5. V. Haarslev and R. Möller. Racer: A core inference engine for the semantic web. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools*, volume 87, 2003.
6. T. Härder and A. Reuter. Principles of Transaction-Oriented Database Recovery. *ACM Comput. Surv.*, 15(4):287–317, December 1983.
7. P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph. OWL 2 web ontology language primer. *W3C recommendation*, 27:1–123, 2009.
8. M. Horridge and S. Bechhofer. The OWL API: a Java API for working with OWL 2 ontologies. *Proc. of OWL Experiences and Directions*, 2009, 2009.
9. J. Lu, L. Ma, L. Zhang, J.-S. Brunner, C. Wang, Y. Pan, and Y. Yu. SOR: a practical system for ontology storage, reasoning and search. In *Proceedings of the 33rd international conference on Very large data bases*, pages 1402–1405. VLDB Endowment, 2007.
10. L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a Complete OWL Ontology Benchmark. In *ESWC*, pages 125–139, 2006.
11. P. McBrien, N. Rizopoulos, and A. Smith. SQOWL: Type Inference in an RDBMS. *Conceptual Modeling–ER 2010*, pages 362–376, 2010.
12. P. McBrien, N. Rizopoulos, and A. Smith. Type inference methods and performance for data in an RDBMS. In *Proceedings of the 4th International Workshop on Semantic Web Information Management*, page 6. ACM, 2012.
13. D. L. McGuinness, F. Van Harmelen, et al. OWL web ontology language overview. *W3C recommendation*, 10(2004-03):10, 2004.
14. Z. Pan, X. Zhang, and J. Heflin. DLDB2: A scalable multi-perspective semantic web repository. In *Web Intelligence*, pages 489–495, 2008.
15. Pellet. http://clarkparsia.com/pellet/.
16. J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan. Minerva: A scalable OWL ontology storage and inference system. In *The Semantic Web–ASWC 2006*, pages 429–443. Springer, 2006.