# 관계형 데이터베이스를 이용한 효율적인 OWL 속성 추론 기법
(An Efficient Reasoning Method for OWL Properties using Relational Databases)

린 제 시 [†] 이 지 현 [†] 정 진 완 [††]
(Jiexi Lin) (Jihyun Lee) (Chin-Wan Chung)

**요 약** OWL(Web Ontology Language)은 시맨틱웹에서 온톨로지를 배포하고 공유하기 위한 W3C의 정식 권고안(Recommendation)으로 채택되었다. OWL 데이터의 숨겨진 정보를 유추하기 위해서 OWL 추론기들이 많이 개발되었다. 그러나 OWL 추론기들은 메모리를 기반으로 처리되기 때문에 대용량 OWL 데이터를 처리하기는 어렵다. 이런 문제를 해결하기 위해서 관계형 데이터베이스에 기반한 시스템이 제안되었다. 이 시스템들은 OWL 데이터를 데이터베이스에 저장하여 데이터베이스 내에서 추론을 한다. 하지만, 이 시스템들은 OWL에서 정의되는 모든 속성(Property)을 고려하지 않았고, 추론에 비효율적인 스키마를 사용하고 있다. 그리고 실제 응용환경에서 자주 발생하는 OWL 데이터 변경에 대해서도 다루지 않았다. 본 논문에서는 관계형 데이터베이스에 기반한 여러 스키마를 비교하고, 효율적인 추론을 위한 개선된 스키마를 제안한다. 그리고 OWL에서 정의되는 모든 종류의 속성을 지원하기 위한 완전하고 효율적인 추론 알고리즘과 OWL 데이터 변경에 대해 효율적인 갱신 방법을 제안한다. 실험결과를 보면 본 논문에서 제안한 스키마가 OWL 데이터 저장 및 추론에 대해 기존 스키마보다 더 좋은 성능을 보이며, OWL 데이터 갱신 방법도 기존의 방법보다 더 효율적이다.

**키워드** : 온톨로지, 시맨틱웹, OWL, 추론

***Abstract*** The Web Ontology Language (OWL) has become the W3C recommendation for publishing and sharing ontologies on the Semantic Web. To derive hidden information from OWL data, a number of OWL reasoners have been proposed. Since OWL reasoners are memory-based, they cannot handle large-sized OWL data. To overcome the scalability problem, RDBMS-based systems have been proposed. These systems store OWL data into a database and perform reasoning by incorporating the use of a database. However, they do not consider complete reasoning on all types of properties defined in OWL and the database schemas they use are ineffective for reasoning. In addition, they do not manage updates to the OWL data which can occur frequently in real applications.

In this paper, we compare various database schemas used by RDBMS-based systems and propose an improved schema for efficient reasoning. Also, to support reasoning for all the types of properties defined in OWL, we propose a complete and efficient reasoning algorithm. Furthermore, we suggest efficient approaches to managing the updates that may occur on OWL data. Experimental results show that our schema has improved performance in OWL data storage and reasoning, and that our approaches to managing updates to OWL data are more efficient than the existing approaches.

**Key words** : Ontology, Semantic Web, OWL, Reasoning

· 이 논문은 2009년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2009-0081365).

† 학생회원 : 한국과학기술원 전산학과
linjiexi@hotmail.com
hyunlee@islab.kaist.ac.kr
†† 정 회 원 : 한국과학기술원 전산학과 교수
chungcw@cs.kaist.ac.kr

논문접수 : 2006년 10월 24일
심사완료 : 2009년 11월 29일

## 1. Introduction

As the next generation of the World Wide Web, the Semantic Web aims at providing machine processable information on the Web. With such a vision, the Web can reach its full potential only if it becomes a place where data can be shared and processed by automated tools as well as by people.

Ontologies are the backbones that provide semantics to the Semantic Web by defining shared and common vocabularies. Tom Gruber gives the formal definition of ontology as "an ontology is a specification of a conceptualization," which is widely accepted by most researchers currently.

For publishing and sharing ontologies on the Web, the Web Ontology Language (OWL) [1] has been proposed by W3C as a recommendation. Since OWL gives well-defined information about data, reasoning can be performed on them and hidden information can be derived. A number of OWL reasoners [2-4] have been proposed to manage OWL data and perform the reasoning. These OWL reasoners perform TBox (assertions on concepts) as well as ABox (assertions on individuals) reasoning. Since OWL reasoners perform reasoning in memory, they cannot handle large-sized OWL data.

To overcome the scalability problem of OWL reasoners, RDBMS-based systems [5-8] have been proposed. These systems store OWL data into a database and perform reasoning on them in the database. However, the database schemas they use are ineffective for reasoning. In addition, they do not consider complete reasoning about all types of properties defined in OWL. Also, they do not manage the changes of OWL data which are happened frequently in real applications.

In this paper, we propose different efficient reasoning techniques of ontologies in relational database. The reasoning scope can be considered as ABox reasoning related to OWL properties. These techniques include an improved database schema design, a complete reasoning algorithm for all types of OWL properties, and efficient approaches for change management of OWL data.

**An Improve property based schema design:** After comparing several schemas that are used by existing RDBMS-based systems, we propose an improve schema based on the property-based schema [9]. We name it the improved property-based schema. With our schema, reasoning for inverseOf properties can be done in memory during the storage of OWL data and no additional change management of inverseOf properties is necessary.

**The ISFT reasoning algorithm:** Since facts derived over the first iteration of reasoning can introduce other new facts, reasoning needs to be performed iteratively until no new facts can be generated. A complete reasoning algorithm for inverseOf, symmetric and transitive properties has been proposed by ONTOMS [8]. It is called the IST reasoning algorithm since it follows the sequence <inverseOf reasoning, symmetric reasoning, transitive reasoning>. To support complete reasoning for all types of OWL properties, we propose the ISFT reasoning algorithm taking functional properties into account. Also, we analyze several possible approaches for functional reasoning and propose a graph based algorithm which we prove to be the most efficient.

**Maintenance of ISFT reasoning:** When updates are applied to OWL data after reasoning, the related changes should be managed to maintain the integrity of the OWL data. For example, the insertion of facts can introduce more new facts, and the deletion of facts can lead to deletion of some derived facts. Update management of transitive reasoning is most expensive because a large number of facts can be generated or deleted. In this paper, we analyze the approaches for update management of OWL data and propose a disjoint graph algorithm which has the best performance in maintenance of transitive reasoning.

The remainder of this paper is organized as follows. Section 2 introduces other works based on RDBMS systems. Section 3 gives the definitions of OWL properties. Section 4 compares different schema design approaches and proposes the improved property schema. With such a schema, Section 5 proposes a complete reasoning algorithm for all types of OWL properties. After discussing the change management of the OWL data in Section 6, the experimental results are shown in Section 7. The conclusion and the future work are presented in Section 8.

## 2. Related Work

To overcome the scalability problem of OWL reasoners, several RDBMS-based systems have been proposed. They store OWL data into a database and perform reasoning on them in the database. These RDBMS based systems include SnoBase [5], the Instance Store [6], DLDB [7] and ONTOMS [8].

SnoBase [5], which has been merged into the IBM Integrated Ontology Development Toolkit (IODT) [10], employs the database schema which uses only one universal relation to store class, property and instance definitions. For ontology with large amount of instances, the univeral relation will be extremely large and thus leads to big scanning overhead. In addition, SnoBose attempts to perform reasoning by incorporating SQL triggers. However, the runtime depth level of trigger cascading supported in RDBMSs is limited, which leads to difficulties or impossibilities in performing reasoning of OWL data.

The Instance Store [6] uses a relation to store class defnitions, another relation to store instances, and four relations (Type, Equivalents, Parents and Children) to maintain class hierarchy information. The Instance Store performs reasoning for instances which do not have properties (referred to as role-free ABox in [6]). Thus, no reasoning for OWL properties is performed. This limitation prevents Instance Store from processing ordinary OWL data.

DLDB [7] maintains one class relation for each class and one property relation for each property so that instances could be scattered into related relations instead of one large relation. This schema is neat and has better performance in reasoning for instances since the length of relations are greatly reduced. DLDB uses FaCT [2], an OWL reasoner, to get complete hierarchy information of class and properties. However, DLDB does not support any instance reasoning for OWL data so that we could not make comparison of reasoning performance with DLDB.

ONTOMS [8] employs a class-based database schema which maintains a relation for each class containing associated properties as its attributes. This schema becomes more complicated than that of previous RDBS-based systems but has better performance in instance reasoning than the schema of DLDB. Detailed analysis of the difference of two schemas and their query processing performance are given in [8]. ONTOMS supports instance reasoning for inverseOf, symmetric and transitive properties (referred to as IST properties in [8]). The instance reasoning algorithm in ONTOMS (referred to as IST reasoning algorithm in [8]) emphasizes on the sequence of performing reasoning so as to avoid iterative processing. However, the IST reasoning algorithm does not consider other two properties defined in OWL, i.e., the functional property and the inverseFunctional property, which can derive new facts from the original OWL data.

A limitation that is shared by all the above mentioned system is the fact that they do not manage updates to OWL data which can occur in real applications.

## 3. OWL and OWL Properties

Web Ontology Language (OWL) [1] is a semantic markup language developed as a vocabulary extension of ontology data. Different from RDF [11], OWL provides more expressive power for describe ontologies such as equality and inequality of classes and properties, cardinality restrictions of properties and five characteristics of properties, i.e., inverseOf property, symmetric property, transitive property, functional property and inverseFunctional property. The definitions of the five types of OWL properties can be given as follows.

*Defnition 1 (inverseOf Property).* Property P is specified as inverseOf property of property P' if for all individuals x and y: P(x,y) implies P'(y,x) where x is related to y by property P.

*Defnition 2 (symmetric Property).* Property P is specified as symmetric if for all individuals x and y: P(x,y) implies P(y,x).

*Defnition 3 (transitive Property).* Property P is specified as transitive if for all individuals x, y and z: P(x,y) and P(y,z) implies P(x,z).

*Defnition 4 (functional Property).* Property P is specified as functional if for all individuals x, y and z: P(x,y) and P(x,z) implies y is equal to z.

*Defnition 5 (inverseFunctional Property).* Property P is specified as an inverseFunctional property which

```
<owl:Class rdf:ID="Professor">
  <rdfs:subClassOf rdf:resource="Person"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="degreeFrom">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#University"/>
  <owl:inverseOf rdf:resource="hasAlumnus"/>
</owl:ObjectProperty>
<Professor rdf:about="Prof1">
  <degreeFrom rdf:about="Univ1"/>
  <memberOf rdf:about="Dept1"/>
  <teachesCourse rdf:about="Course1"/>
  <teachesCourse rdf:about="Course2"/>
</Professor>
```

Figure 1 An example of OWL document

is inverseOf property P' if for all individuals x, y and z: P(y,x) and P(z,x) implies y is equal to z.

Figure 1 presents an example of an OWL document (which is also referred to as OWL data). In Figure 1, the *degreeFrom* property is defined as inverseOf the *hasAlumnus* property, and *Prof1* has a degree from *Univ1*. Apparently, after reasoning, the fact that *Univ1* has *Prof1* as an alumnus should be derived.

## 4. Database Schema Design

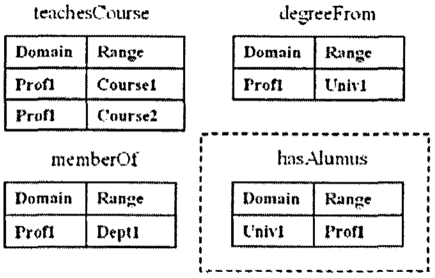
To load the OWL data into a database and perform reasoning, there are several optional schema designs. After analyzing several database schemas, we propose an improved schema design which is proved to be more efficient in reasoning of OWL properties.

SnoBase [5] uses an instance based schema that has only one relation. The relation stores class, property and instance definitions. This schema is simple and allows OWL data to be easily parsed and loaded into the database. However, because it only uses one universal relation, the relation may be of extremely large size which may greatly deteriorate the performance of reasoning.

Another alternative schema is to create a relation for each class, i.e., the class relation. Currently, this schema is employed by ONTOMS [8]. The class relation contains associated properties as its attributes. This may correspond to the entity-relational approach frequently used when design database schemas. When a property is multi-valued, a separate class-property relation is assigned to the specific class and property. This schema requires higher storage cost because the OWL data have to be arranged to be put into the corresponding column. Although the reasoning cost may not be as great as that of the instance-based schema, the class-based schema does have difficulties in reasoning. For example, when storing a newly derived fact (triple) into the database, we have to check the type of the subject instance first in order to add it to the corresponding relation. Checking the type of an instance can be very expensive if there are a large number of instances. As a result, reasoning with such a schema is not straightforward.

A property-based schema is proposed which takes into account the performance of both query evaluation and reasoning [9]. The property schema creates a relation for each property. The property relation stores the (subject, object) pairs of a property together with the domain type and the range type of a subject/object. With such a schema, we do not need to check the type of the subject in a derived fact before inserting it into the database, and the reasoning for OWL properties becomes much more straightforward.

Although the property-based schema is efficient and straightforward in reasoning for OWL properties, it leads to redundant entries in the database. To overcome this problem, we propose an improved property-based schema which only keeps one relation for two properties that are inverseOf each other. Figure 2 shows the imporved property-based relations. For example, when parsing the OWL data, if we get the fact hasAlumnus(Univ1, Prof1), we simply treat it as the fact degreeFrom(Prof1, Univ1) and store it into the degreeFrom table. Thus, table hasAlumnus is not necessary to be generated. As a


teachesCourse

| Domain | Range |
|---|---|
| Prof1 | Course1 |
| Prof1 | Course2 |

degreeFrom

| Domain | Range |
|---|---|
| Prof1 | Univ1 |

memberOf

| Domain | Range |
|---|---|
| Prof1 | Dept1 |

hasAlumnus

| Domain | Range |
|---|---|
| Univ1 | Prof1 |

Figure 2 Improved Property-Based Relations

result, reasoning for inverseOf properties is performed in memory during the storage of OWL data. This leads to much more efficient reasoning for three reasons. First, reasoning in memory is much faster than in a database. Second, inverseOf reasoning is only performed once. Another advantage is that maintenance after data changes can be performed more easily.

## 5. Reasoning for OWL Properties

To provide enhanced reasoning about a property, OWL defines five types of properties: inverseOf, symmetric, transitive, functional and inverseFunctional. According to the characteristic of each property, new facts can be derived. This Section gives the definitions of OWL property reasoning, analyzes the reasoning tasks and proposes an efficient algorithm for functional algorithm. Finally, we propose a complete reasoning algorithm for the five types of properties and give the proof to its completeness. To our knowledge, no existing RDBMS-based systems handle reasoning for these five properties.

### 5.1 Definitions of OWL Property Reasoning

Since the OWL property reasoning mentioned in this paper is based on relation database, all the reasoning is performed with database relations. Definition 6 states the meaning for a relation of a property.

*Defnition 6 (Relation of a property).* Relation R of a property P is the set of (x,y) in P.

According to the definitions of OWL properties given in Section3, the defitions of inverseOf, symmetric, transitive, functional and inverseFunctional reasoning for property P or the relation of P are given below.

*Defnition 7 (inverseOf reasoning).* Let property P be inverseOf property P′, R be the relation of P, and R′ be the relation of P′. The inverseOf reasoning for P or R is the process of adding (y,x) to R for all (x,y) in R′, if (y,x) is not in R. [8]

*Defnition 8 (symmetric reasoning).* Let P be a symmetric property and R be the relation of P. The symmetric reasoning for P or R is the process of adding (y,x) to R for all (x,y) in R, if (y,x) is not in R. [8]

*Defnition 9 (transitive reasoning).* Let P be a transitive property and R be the relation of P. The transitive reasoning for P or R is the process of computing the transitive closure of R. [8]

*Defnition 10 (functional reasoning).* Let P be a functional property and R be the relation of P. The funtional reasoning for P or R is the process of finding the fact y=z if (x,y) and (x,z) are in R.

*Defnition 11 (inverseFunctional reasoning).* Let P be an inverseFunctional property which is inverseOf property P′, R be the relation of P, and R′ be the relation of P′. The inverseFuntional reasoning for P or R is the process of finding the fact y=z if (y,x) and (z,x) are in R'.

To simplify the reasoning procedures, we introduce Lemma 1 which states that symmetric property also possesses the characteristic of inverseOf property. Therefore, symmetric reasoning can be treated as a special case of inverseOf reasoning and the algorithms for symmetric reasoning and inverseOf reasoning are similar.

*Lemma 1.* Symmetric property can be seen as inverseOf itself.

As far as transitive reasoning is concerned, we use a graph based algorithm [12] with complexity O(V(V+E)) to compute the transitive closure of a relation. We first construct a graph of every (x,y) in R where in the graph, x and y are vertices and x->y is an edge. Then we compute the transitive closure of the graph in memory and insert the newly generated edges into the database as new facts.

Since an inverseFunctional property P is inverseOf another property P' which is a functional property, invereseFunctional reasoning for P can be divided into two steps. First, perform inverseOf reasoning for P. Second, perform functional reasoning for P'. Notice these two steps of reasoning can be performed respectively. As a result, if P is marked as inverseOf P' and P' is marked as functional, we can omit the inverseFunctional reasoning for P. We do not consider inverseFunctional reasoning in later sections in this paper.

According to Definition8, equal instances can be found using the characteristic of the functional property. Consequently, the functional reasoning triggers the sameAS reasoning which is defined in Definiton12.

*Defnition 12 (sameAs reasoning).* Let a and b be two equal instances. The sameAs reasoning for (a=b) is the process: adding (b,x) for all (a,x) if (b,x) is not in the database; adding (x,b) for all (x,a) if (x,b) is not in the database; adding (a,y) for all (b,y) is (a,y) is not in the database and adding (y,a) for all (y,b) if (y,a) is not in the database where x and y are any instances.

Finally, we consider the algorithm for functional reasoning. Since functional reasoning generates a set of equal instances, sameAs reasoning needs to be performed. However, since sameAs reasoning can generate more new facts which leads to iterative reasoning until no more facts can be generated. Section5.2 discusses the algorithms for functional reasoning.

### 5.2 An Efficient Functional Reasoning Algorithm

As can be seen from the definitions of OWL property reasoning, functional reasoning causes the sameAs reasoning to be performed. This leads to iterative reasoning. In this section, we first analyze a naïve approach and a direct approach for functional reasoning. Last, we propose an efficient graph-based reasoning algorithm.

#### 5.2.1 A naïve functional reasoning algorithm

A naïve functional reasoning algorithm performs the reasoning in the sequence <Functional reasoning, sameAs reasoning, Functional reasoning, sameAs reasoning…> until there are no new facts that are generated. Figure 3 shows the reasoning procedures. Relation R is the original relation defined as functional. After functional reasoning is performed for the first time, we get the equal instance set S: (b=c), (e=f) and (a=d). Then we perform sameAs reasoning to obtain relation R'. Next, functional reasoning is performed for the second time to obtain the equal instance set S'. Reasoning is performed iteratively until there are no new facts that can be generated. The disadvantage of the naïve approach lies in that duplicated facts are derived frequently (see the dotted part in Figure 3) which can be seen as unnecessary reasoning.

#### 5.2.2 A direct functional reasoning algorithm

A direct functional reasoning algorithm performs the functional reasoning iteratively until no new equal instances can be found, and finally performs the sameAs reasoning. It finds the equal instance sets by using a matching method such as an SQL (as shown in Figure 5). This approach avoids generating duplicated facts. Figure 4 shows the reasoning steps.

#### 5.2.3 A graph based functional reasoning algorithm

A graph based functional reasoning algorithm builds a directed graph according to the relation of the functional property and gets the complete set of equal instances. Like the direct algorithm, sameAs reasoning is performed last. This algorithm builds a graph consisting of tuples in the functional relation. For example, for tuple (a,b) in relation R, a and b are vertices and a->b is an edge in the graph.

With such a graph, finding the equal instance sets can be achieved by merging the child nodes of the node whose out-degree is bigger than 1. When

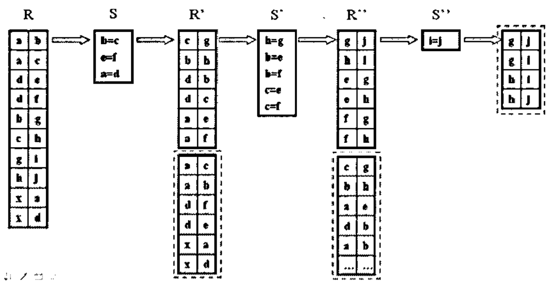


Figure 3 The reasoning procedures of the naïve algorithm

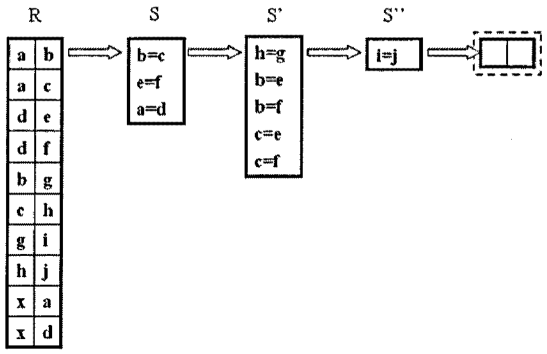


Figure 4 The reasoning procedures of the direct algorithm

```
select R1.u2, R2.u2 from R R1, R R2, S
where R1.u1=S.s1 and R2.u1=S.s2
```

Figure 5 The matching method described in SQL

```
Algorithm Graph_Based (Relation R)
begin
1. Construct Graph g corresponding to Relation R
2. boolean foundAll := false;
3. while (!foundAll) do
4.     for every node Ni in g do
5.         if outDegreeOf(Ni) > 1 then
6.             Merger the children of Ni;
7.             Put the children nodes to the equal set
8.             hashtable;
9.     foundAll = true;
10.    for every node Ni in g do
11.        if outDegreeOf(Ni) > 1 then
12.            foundAll := false;
13.            break;
end
```

Figure 6 The graph based algorithm

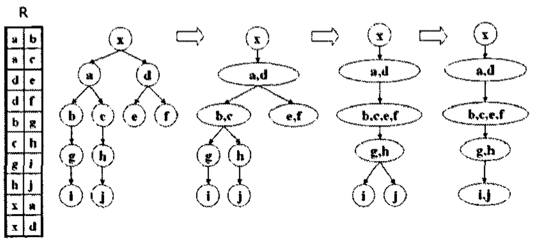


Figure 7 The procedures of finding equal instance sets in the graph based algorithm

the graph reaches a final state, the out-degree of every node should be less than or equal to 1. Figure 6 shows the algorithm and Figure 7 shows the merging procedure. The complexity of the graph based algorithm is O(nm) where n is the number of nodes and m is the number of merging steps. The value of m depends on the structure of the graph and cannot be expected. Usually m is much smaller than n. Even so, the complexity in worst case will be $O(n^2)$.

## 5.3 An Algorithm for Complete OWL Property Reasoning

An OWL property can be seen as a subset of the rule that introduce new facts to the ontology. In order to use such rules in databases, active database systems [13] have been developed. They use forward chaining [14] to iteratively or recursively compute new facts until there are no additional facts that can be generated. To avoid iterative reasoning, ONTOMS [8] proposed an IST reasoning algorithm for inverseOf, symmetric and transitive properties and proved its completeness. Theorem1 provides the basis in proving the completeness of the IST reasoning algorithm.

*Theorem 1.* Suppose property P is inverseOf property P'. By following the sequence <inverseOf reasoning, symmetric reasoning, transitive reasoning> for P and P', the resulting relations of P and P' are inverseOf each other, symmetric and transitive [8].

According to Theorem 1, if a property P is inverseOf property P′, P is symmetric and transitive, then after performing IST reasoning for property P and P′, the relation of P is symmetric, transitive and inverseOf the relation of P′. Thus, no more facts can be generated by performing any inverseOf, symmetric or transitive reasoning for P, which means the reasoning for P is complete.

However, when functional and inverseFunctional properties are taken into account, sameAs reasoning can also be aroused. We have to find out when the sameAs reasoning should be performed in order to avoid iterative reasoning. Within the improved property based schema, it is straightforward that it makes no difference whether sameAs reasoning is performed before or after the inverseOf and symmetric reasoning. However, sameAs reasoning must be performed *before* transitive reasoning.

Figure 8 shows an example in which sameAs reasoning is performed before transitive reasoning. In Figure 8, R is the original relation, (b=d) and (f=c) are information about equal instances. After sameAs reasoning, new facts (a,d), (b,e) and (x,c) are added to R such that R becomes R'. After transitive reasoning for R', we get R". If sameAs reasoning is performed *after* transitive reasoning, we cannot get the fact (a,e) in R". Therefore, sameAs reasoning must performed before transitive reasoning.

The <IST> reasoning sequence suggested by the IST reasoning algorithm successfully solved the iterative and recursive computation problem. Inspired by the IST reasoning algorithm and the observation that functional reasoning which causes sameAs reasoning must be done before the transitive reasoning, we propose an ISFT reasoning algorithm which supports complete reasoning for all types of OWL properties. Theorem2 describes the sequence of ISFT reasoning.

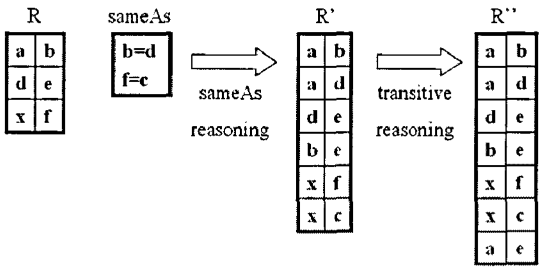


Figure 8 sameAs reasoning performed before transitive reasoning

*Theorem 2.* With respect to the improved property-based schema, a complete set of new facts can be obtained by following the sequence <inverseOf reasoning, symmetric reasoning, Functional reasoning, transitive reasoning>.

***Proof of Theorem2***

According to Theorem1, reasoning following the sequence <IST> is complete. With respect to the improved property schema, sameAs reasoning, which is caused by functional reasoning, can be done before or after the IS reasoning. After sameAs reasoning, the result set of transitive reasoning is complete. Thus, the reasoning by following the sequence <ISFT> is complete.

## 6. Maintenance of ISFT Reasoning

In real applications, OWL data undergo frequent updates including insertions, deletions and updates of instances. After such changes, additional reasoning is required to maintain the integrity of the OWL data. For example, new facts may be generated due to the insertion of a fact and some derived facts may be deleted due to the deletion of the original fact. One approach to manage the changes is to parse the OWL data again, reload them into the database and perform reasoning again. However, if OWL data are of large sizes, it may take a long time to do so. An alternative approach is to make small adjustments of the database according to the updates. This section discusses how the latter approach manages the changes of OWL data.

Referring to the schemas other than the improved property-based schema, after changes of OWL data, additional management for inverseOf and symmetric reasoning is required. For example, if the fact degreeFrom(Prof1, Univ1) is delete, the fact hasAlumnus (Univ1, Prof1) should also be deleted. However, with respect to the improved property-based schema, no extra management is needed for inverseOf and symmetric reasoning. Insertion, deletion and updates can be directly performed in the corresponding relation.

Transitive reasoning is the process of computing the transitive closure of a relation. Since maintenance in transitive reasoning is the most expensive, we mainly focus on discussing the maintenance of transitive reasoning. We compare two existing algorithms of maintaining the transitive closure and propose a disjoint graphs algorithm which is suitable for case that the graph of a relation can be divided into many disjoint graphs.

The first algorithm is a naïve memory based algorithm that reconstructs the graph of the whole relation and computes the transitive closure of the graph in memory. Newly generated facts are put back into the database.

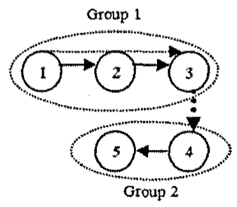
Another algorithm is a database based algorithm that uses pure SQL to maintain the transitive closure. This algorithm was proposed in [15]. Since the algorithm is performed in a database, frequent use of relation joins is unavoidable.

After examining the characteristics of ontology data, we find that the graph of a transitive relation can be usually divided in to smaller disjoint groups. Based on such an observation, we design a disjoint graph algorithm for the maintenance of the transitive closure. Using union find algorithm, the graph can be divided into disjoint graph groups at the beginning and each group can be marked with a group ID (GID). The group ID is maintained after updates. Notice that after edge insertion, two groups can be merged and after edge deletion, a group can be split into small groups.

Figure 9 shows an example of edge insertion and Figure 10 shows the algorithm. In Figure 9, when edge (3,4) is inserted, it affects two groups, i.e., Group1 and Group2, but not the Group3. Thus, Group3 does not need to be brought to memory for graph constructing. Also, Group3 does not affect the computation of the transitive closure of Group1 and Group2. After computation, Group1 and Group2 are merged into one group.

| N1 | N2 | GID |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 3 | 1 |
| 1 | 3 | 1 |
| 4 | 5 | 2 |
| 10 | 11 | 3 |
| 11 | 12 | 3 |
| 10 | 12 | 3 |
| 3 | 4 | |

| N1 | N2 | GID |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 3 | 1 |
| 1 | 3 | 1 |
| 4 | 5 | 1 |
| 10 | 11 | 3 |
| 11 | 12 | 3 |
| 10 | 12 | 3 |
| 3 | 4 | 1 |
| 1 | 4 | 1 |
| 1 | 5 | 1 |
| 2 | 4 | 1 |
| 2 | 5 | 1 |
| 3 | 5 | 1 |



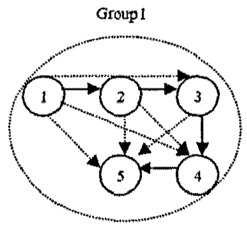
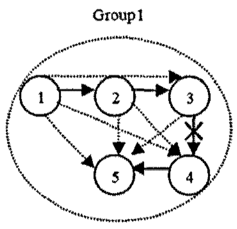


Figure 9 Transitive closure maintenance after edge insertion

| N1 | N2 | GID |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 3 | 1 |
| 1 | 3 | 1 |
| 4 | 5 | 1 |
| 10 | 11 | 3 |
| 11 | 12 | 3 |
| 10 | 12 | 3 |
| X 3 | 4 | 1 |
| 1 | 4 | 1 |
| 1 | 5 | 1 |
| 2 | 4 | 1 |
| 2 | 5 | 1 |
| 3 | 5 | 1 |

| N1 | N2 | GID |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 3 | 1 |
| 1 | 3 | 1 |
| 4 | 5 | 2 |
| 10 | 11 | 3 |
| 11 | 12 | 3 |
| 10 | 12 | 3 |



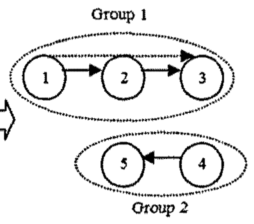


Figure 11 Transitive closure maintenance after edge deletion

```
Algorithm Disjoint_Graphs_insertion (Relation R, (a,b))
/* (a,b) is the tuple to be inserted into Relation R */
begin
1. int g1, g2;
2. relation t, t';
3. g1 := select (first_row) GID from R where n2 = a;
   //find a's group
4. g2 := select (first_row) GID from R where n1 = b;
   //find b's group
5. t := select n1,n2 from R where GID=g1 or GID=g2
6. Compute the transtive closure of t and get new
   generated relation t';
   //perform transitive reasoning with only two groups
7. insert (t', g1) into R;
8. update R set GID = g1 where GID = g2;
   //maintain the group id
end
```

Figure 10 Disjoint Graph Algorithm After Insertion

An example of the edge deletion is shown in Figure 11. When the edge (3,4) is deleted, only Group1 is affected. Group3 does not need to be considered during transitive closure maintenance. After transitive closure computation Group1 is split into two small groups. The algorithm is similar with that of insertion.

The disjoint graphs algorithm saves the cost of conveying unaffected groups into memory, and the computing of their transitive closures.

## 7. Experimentation

### 7.1 Experimental Environment

The implementation of the algorithm are written in Java. We use OWL API Parser to parse the OWL data and IBM DB2 UDB 8.2 as an RDBMS. Our experiment was performed on a Windows XP system running a 3GHz Pentium 4 processor with 512MB of main memory.

Three data sets are used for experiment. The first data set DS1 is the University Benchmark Data (LUBM) [16] which is a widely used OWL benchmark data set. LUBM not only generates data with arbitrary size but also has a complex data schema. The ontology of LUBM is about universities. It contains inverseOf properties and transitive properties. Therefore, LUBM is appropriate for evaluating the instance reasoning. We generated various sizes of OWL data: 1MB, 5MB, 10MB, 50MB, 100MB, and 500MB.

The second data set DS2 is a synthetic data set designed to test the performance of the functional reasoning with respect to the three algorithms proposed in Section 5. The data set contains pairs (1,2), (1,3), (2,4), (2,5), (3,6), (4,7), (5,8), (7,9), (8,10) ···, which requires at least four times of iteration to get the complete equal instance sets.

The last data set DS3 is a synthetic data set used to test the performance of maintenance of transitive closure. The data set consists of pairs (1,2), (2,3), ··· , (n-1, n), (n+1, n+2), (n+2, n+3), ··· where (n,n+1) can be inserted to test updates.

DS2 and DS3 are directly generated and inserted into the database for testing instead of retrieving from the OWL ontology. DS2 can have arbitrary vertices and edges. Also DS3 can be generated with arbitrary depth.

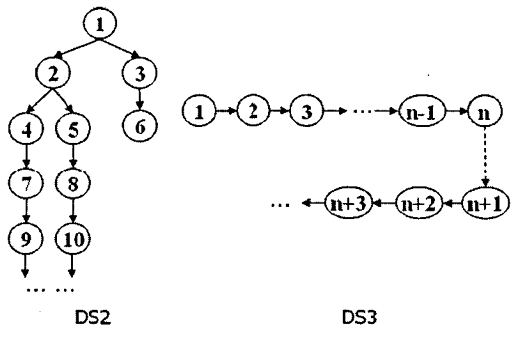


Figure 12 Structure Of Ds2 And Ds3

## 7.2 Storage and Reasoning Time

We tested the storage time and the reasoning time for the class-based schema and the improved property-based schema with data set DS1 of various sizes.

Figure 13 shows the comparison result of storage time between the class-based schema and the improved property-based schema. It is evident the improved property based schema outperforms the class based schema. This can be attributed to the fact that data parsed from an OWL document must be arranged into the corresponding columns in the class relation when using the class based schema.

As stated in Section 4, inverseOf and symmetric reasoning is performed during storage time when using the improved property based• schema. Thus, additional inverseOf and symmetric reasoning is not required after the OWL data are stored into the database. Figure 14 compares reasoning time.

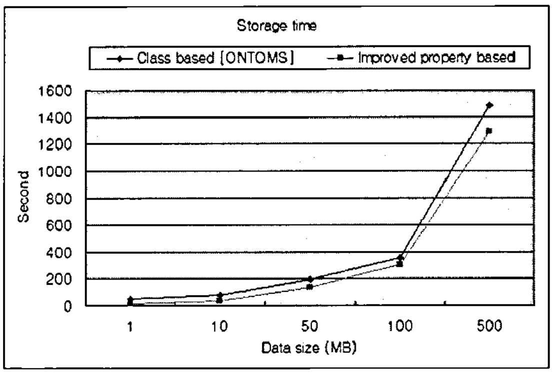


Figure 13 Storage time of class based schema and improved property based schema

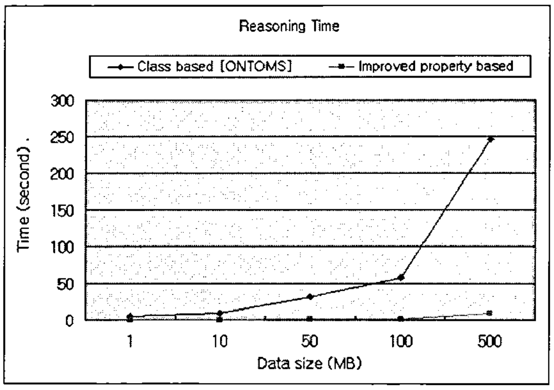


Figure 14 Reasoning time of class based schema and improved property based schema

## 7.3 Functional Reasoning Time

As described in Section 5, we have proposed three algorithms for functional reasoning. Figure 15 shows the performance of the three algorithms with data set DS2. Since the naïve algorithm performs redundant reasoning, its performance deteriorates rapidly as the data size increases. The direct algorithm outperforms the naïve algorithm by avoiding the redundant reasoning. However, the matching algorithm to find the equal instance sets iteratively still remains inefficient. Compared with the former two algorithms, the graph based algorithm has the best performance. With data set DS2, the reasoning time of the graph based algorithm increases linearly.

## 7.4 Transitive Closure Maintenance Time

We tested the performance of transitive closure maintenance among the naïve memory based algorithm, the DB based algorithm and the graph based

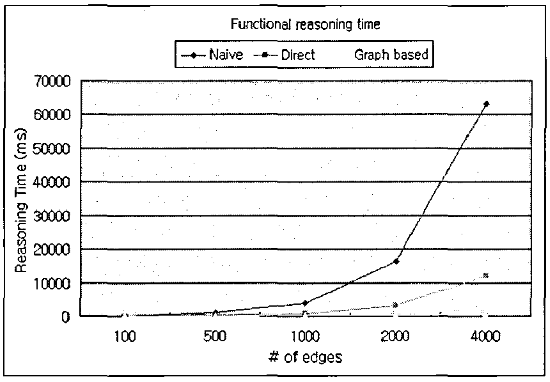


Figure 15 Functional Reasoning time with data set DS2

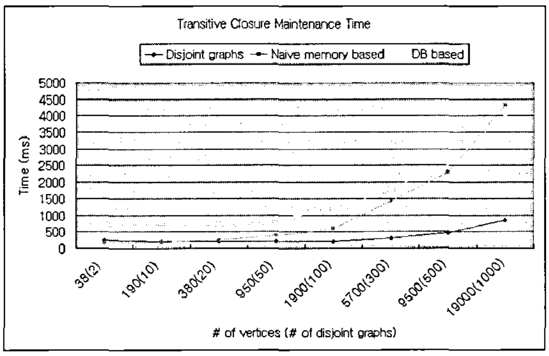


Figure 16 Transitive closure maintenance time with data set DS3

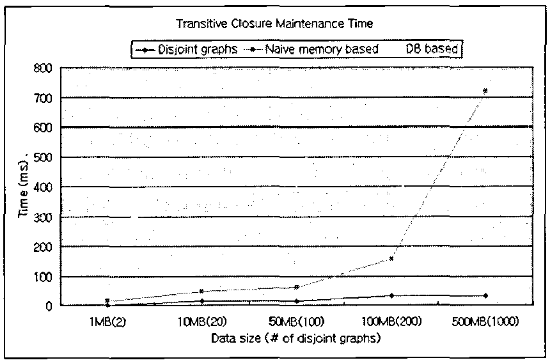


Figure 17 Transitive closure maintenance time with data set DS1

algorithm. Figure 16 and Figure 17 illustrate the performance of the transitive closure of a relation with data set DS3 and DS1, respectively.

Due to the poor performance of relational joins, the DB based algorithm has worse performance than the naïve memory based algorithm when the number of disjoint graphs is not large. However, with a large number of disjoint graphs, the naïve memory based algorithm is less efficient because it re-calculates the transitive closure of the whole graph. In the 100MB data set from DS1, there are 200 universities defined, thus forming 200 disjoint graphs. In the same way, we can construct 1000 disjoint graphs in 500MB sized data set DS1. With such a large number of disjoint graphs, the disjoint graphs algorithm performs much better than the naïve memory based algorithm and the DB based algorithm.

## 8. Conclusion and Future Work

There exist various RDBMS-based systems for storage and reasoning over OWL data. Since these systems use ineffective database schemas, they exhibit deteriorated reasoning performance. Also, they do not support complete reasoning for all types of OWL properties or update management of OWL data.

To solve these problems, this paper proposes efficient techniques for reasoning over ontologies in a relational database. These techniques include an improved database schema design, a complete reasoning algorithm for all types of OWL properties, and an efficient approach for managing updates to OWL data. Experimental results show that the improved database schema is more efficient than the existing database schema and that our approach for change management of OWL data is efficient than existing approaches.

Currently, the management of ontologies is performed in a single database. For our future work, we plan to focus on the research of reasoning for distributed ontologies in a distributed environment.

## References

[1] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. W3C Recommendation, http://www.w3.org/TR/owl-ref, February 2004.

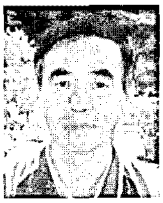
[2] I. Horrocks and P. F. Pastel-Schneider. FaCT and DLP. In *Proceedings of International Conference on Analytic Tableaux and Related Methods*, pp. 27-30, May 1998.

[3] V. Haarsley and R. Moller. RACER System

Description. In *Proceedings of 1st International Joint Conference on Automated Reasoning*, pp. 701-706, June 2001.
[4] B. Parsia and E. Sirin. Pellet: An OWL DL Reasoner. In *Proceedings of 3rd International Semantic Web Conference*, November 2004.
[5] J. Lee and R. Goodwin. Ontology Management for Large-Scale Enterprise Systems. IBM Technical Report, RC23730, September 2005.
[6] I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The Instance Store: Description Logic Reasoning with Large Numbers of Individuals. In *Proceedings of 2004 International Workshop on Description Logic*, pp.31-40, June 2004.
[7] Z. Pan and J. Hefflin. DLDB: Extending Relational Databases to Support Semantic Web Queries. In *Workshop on Practical and Scaleable Semantic Web Systems, ISWC 2003*, pp.109-113, November 2003.
[8] Myung-Jae Park, Ji-Hyun Lee, Chun-Hee Lee, Jiexi Lin, Olivier Serres and Chin-Wan Chung. ONTOMS: An Efficient and Scalable Ontology Management System. Technical Report CS-TR-2005-246, Department of Computer Science, KAIST, December 9, 2005.
[9] Myung-Jae Park and Chin-Wan Chung. Property Based OWL Storage Schema in Relational Databases. Technical Report CS-TR-2005-247, Department of Computer Science, KAIST, December 13, 2005.
[10] IBM Integrated Ontology Development Toolkit, http://www.alphaworks.ibm.com/tech/semanticstk
[11] Brian McBride, Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF) Concepts and Abstract Syntax. W3C Recommendation 10 February 2004.
[12] S. S. Skiena. The Algorithm Design Manual. Telos/Springer-Verlag Publisher, November 1997.
[13] N. W. Paton. Active Database Systems. ACM Computing Surveys, 31(2):63-103, March 1999.
[14] S. Russell and P. Norvig. Artifcial Intelligence-A Modern Approach. Prentice Hall, November 2003.
[15] Guozhu Dong and Jianwen Su. Incremental Maintenance of Recursive Views Using Relational Calculus/SQL. Sigmod Record, 29(1):44-51, March 2000.
[16] Y. Guo, Z. Pan, and J. Hefflin. An Evaluation of Knowledge Base Systems for Large OWL Datasets. In *Proceedings of 3rd International Semantic Web Conference*, pp.274-288, November 2004.

Jiexi Lin
2001년 중국 심천대학교 컴퓨터공학과(학사). 2006년 한국과학기술원 전산학과(석사). 관심분야는 시맨틱 웹, 온톨로지 데이터 관리, 웹 정보검색

이 지 현
2002년 숭실대학교 컴퓨터공학부(학사) 2004년 한국과학기술원 전산학과(석사) 2010년 한국과학기술원 전산학과(박사). 현재 한국과학기술원 전산학과 박사후과정. 관심분야는 시맨틱 웹, 정보검색, 소셜네트워크, 모바일 웹, XML, OWL



정 진 완
1973년 서울대학교 공과대학 전기공학과 학사. 1983년 University of Michigan컴퓨터공학과 박사. 1983년~1993년 미국 GM 연구소 전산과학과 선임연구원 및 책임연구원. 1993년~현재 한국과학기술원 전산학전공 부교수 및 교수. 관심분야는 시맨틱 웹, 소셜네트워크, 모바일 웹, XML, 멀티미디어 데이타베이스, 스트림 데이타 및 센서 네트워크 데이타베이스