An Optimization for Query Answering on $\mathcal{ALC}$ Database

Pakornpong Pothipruk **Guido Governatori**

School of Information Technology and Electrical Engineering
University of Queensland, Australia 4072,
Email: pkp@itee.uq.edu.au

Abstract

Query answering over OWLs and RDFs on the Semantic Web is, in general, a deductive process. To this end, OWL, a family of web ontology languages based on description logic, has been proposed as the language for the Semantic Web. However, reasoning even on $\mathcal{ALC}$, a description logic weaker than OWL, faces efficiency problem. To obviate this problem, at least for $\mathcal{ALC}$, we propose a partition approach that improves the efficiency by splitting the search space into independent Aboxes. Each partition class, i.e., an Abox, can be queried independently. The answer to a query is the simple combination of the answers from each Abox. We prove the correctness of this approach and we outline how to represent compactly the content of each independent Abox. This work can be seen as an optimization for querying a deductive semi-structured database.

Keywords: Description Logic, Query Optimization, Web Database.

1 Introduction

The *Semantic Web*, originated from an idea of the creator of the Web Tim Berners-lee (Berners-Lee 1999), is an effort to bring back structure to information available on the Web. The structures are semantic annotations that conform to an explicit specification (called ontology) of the intended meaning of a piece of information. Thus the the Semantic Web contains implicit knowledge, and information on the Semantic Web is often incomplete since it assumes open-world semantics. In this perspective query answering on the Semantic Web is a deductive process (Stuckenschmidt 2003).

RDF, a semi-structure data page, is a basic component for the Semantic Web. Thus, in the Semantic Web perspective, there are huge number of RDF data pages. In addition, A family of web ontology languages (OWL) based on *Description Logic* (DL) has been proposed as the languages to represent and reason with the Semantic Web. In the nutshell, querying the Semantic Web is the process of reasoning over OWLs and RDFs, based on DL reasoning. DL emphasizes clear unambiguous languages supported by complete denotational semantics and tractable/intractable reasoning algorithms (McGuinness, Fikes, Stein & Hendler 2003). Nevertheless, DL still faces problems when applied in context of the Web. One of them is the efficiency of query answering. Consequently, there is an urgent need of optimizations for querying the Semantic Web.

There are many works about DL reasoning optimization. However, most of them focus only on ontology reasoning with out any data, i.e., DL-Tbox reasoning. In fact, DL-Abox reasoning, which is the basis for querying OWL and RDF on the Semantic Web, was seriously studied by some researchers recently. At present, there are only two prominent works for DL-Abox reasoning optimization, i.e., Instance Store (Horrocks, Li, Turi & Bechhofer 2004) and RACER (Haarslev & Möller 2002). Nevertheless, none of above works can eliminate a chunk of individuals at-a-time from retrieval reasoning. Thus, we create optimization techniques that support this idea. We present here a novel optimization technique for instance checking, an Abox reasoning. We also introduce a technique of instance retrieval, another Abox reasoning.

2 Preliminary: Description Logic

The Semantic Web community implicitly adopted DL as a core technology for the ontology layer. One of the reasons behind this is that this logic have been heavily analyzed in order to understand how constructors interact and combine to affect tractable reasoning, see (Donini, Lenzerini, Nardi & Nutt 1991). Technically, we can view the current Semantic Web, not including rule, proof and trust layers, as a DL knowledge base. Thus, answering a query posed on the Semantic Web (RDF and ontology layers) can be reduced to answering a query posed on a DL knowledge base, not taking into account low-level operations, such as name space resolution.

Description logic itself can be categorized into many different logics, distinguished by the set of constructors they provide. We focus on $\mathcal{ALC}$ description logic since it is the basis of many DL systems.

The language of $\mathcal{ALC}$ consists of an alphabet of distinct concept names **CN**, role names **RN**, and individual names **IN**, together with a set of constructors for building concept and role expressions (Tessaris 2001).

Formally, a description logic knowledge base is a pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is a Tbox, and $\mathcal{A}$ is an Abox. The Tbox contains a finite set of axiom assertions. Axiom assertions are of the form

$$C \sqsubseteq D \mid C \doteq D,$$

where C and D are concept expressions. Concept expressions are of the form

$$A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C,$$

where A is an atomic concept or concept name in **CN**, R is a role name in **RN**, $\top$ (top or full domain) is the most general concept, and $\bot$ (bottom or empty set) is the least general concepts. The Abox contains a finite set of assertions about individuals of the form $a : C$ or $C(a)$ (concept membership assertion) and $(a,b) : R$ or $R(a,b)$ (role membership assertion), where a, b are names in **IN**.

The semantics of description logic are defined in terms of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \bullet^{\mathcal{I}})$, consisting of a nonempty domain $\Delta^{\mathcal{I}}$ and a interpretation function $\bullet^{\mathcal{I}}$. The interpretation function maps concept names into subsets of the domain ($A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$), role names into subsets of the Cartesian product of the domain ($R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$), and individual names into elements of the domain. The only restriction on the interpretations is the so called unique name assumption (UNA), which imposes that different individual names must be mapped into distinct elements of the domain. Given a concept name A (or role name R), the set denoted by $A^{\mathcal{I}}$ (or $R^{\mathcal{I}}$) is called the *interpretation*, or *extension*, of A (or R) with respect to $\mathcal{I}$.

The interpretation is extended to cover concepts built from negation ($\neg$), conjunction ($\sqcap$), disjunction ($\sqcup$), existential quantification ($\exists R.C$) and universal quantification ($\forall R.C$) as follows:

$$
\begin{aligned}
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &= \left\{ x \in \Delta^{\mathcal{I}} | \exists y. \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}} \right\} \\
(\forall R.C)^{\mathcal{I}} &= \left\{ x \in \Delta^{\mathcal{I}} | \forall y. \langle x, y \rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}} \right\}
\end{aligned}
$$

An interpretation $\mathcal{I}$ satisfies (entails) an inclusion axiom $C \sqsubseteq D$ (written $\mathcal{I} \models C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it satisfies an equality $C \doteq D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$. It satisfies a Tbox $\mathcal{T}$ if it satisfies each assertion in $\mathcal{T}$. The interpretation $\mathcal{I}$ satisfies a concept membership assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and satisfies a role membership assertion $R(a,b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. $\mathcal{I}$ satisfies an Abox $\mathcal{A}$ (written $\mathcal{I} \models \mathcal{A}$) if it satisfies each assertion in $\mathcal{A}$. If $\mathcal{I}$ satisfies an axiom (or a set of axioms), then we say that it is a *model* of the axiom (or the set of axioms). Two axioms (or two sets of axioms) are *equivalent* if they have the same models. Given a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ we will say that the knowledge bases entails an assertion α (written $\mathcal{K} \models \alpha$) iff for every interpretation $\mathcal{I}$, if $\mathcal{I} \models \mathcal{A}$ and $\mathcal{I} \models \mathcal{T}$, then $\mathcal{I} \models \alpha$.

The DL Abox can be viewed as a semi-structured database, consisting of a collection of RDF data, while the DL Tbox contains a set of constraints for the data in Abox. Thus, the Tbox can be compared to data modeling in database, e.g., Entity-Relationship data model. However, the semantics of description logics are defined in terms of an interpretation which differentiate description logics from databases. In addition, the domain of interpretation can be chosen arbitrarily, and it can be infinite. The non-finiteness of the domain and the open-world assumption are distinguishing features of description logics with respect to the modeling languages in the database. Even description logics, as modeling languages, overlap to a large extent with modeling languages in database (Baader, Calvanese, McGuinness, Nardi & Patel-Schneider 2003), the particular feature of description logics is in the reasoning capabilities that are associated with it. In other words, while modeling has general significance, the capability of exploiting the description of the model to draw conclusions about the problem at hand is a particular advantage of modeling using description logics. In the next section, we introduce basic reasoning tasks in description logics.

2.1 Reasoning in Description Logic

A description logic knowledge base basically supports two major kinds of reasoning tasks, i.e., Tbox reasoning (ontology reasoning), and Abox reasoning (data reasoning with ontology). Note that, both kinds of reasonings are based on satisfiability problem. Also, keep in mind that, like other logic system, the knowledge base contains implicit knowledge that can be made explicit through inferences.

2.1.1 Reasoning for Tbox

In description logic, basic reasoning services for concepts in Tbox $\mathcal{T}$ consist of (Baader et al. 2003):

- Knowledge base consistency: a Tbox $\mathcal{T}$ is consistent iff it is satisfiable, i.e., there is at least a non empty model for $\mathcal{T}$. An interpretation $\mathcal{I}$ is a model for $\mathcal{T}$ if it satisfies every assertion in $\mathcal{T}$.
- Satisfiability: a concept C is satisfiable with respect to $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is nonempty. $\mathcal{I}$ is also a model of C.

  Definition 1 ***(Satisfiability w.r.t. Knowledge base)*** *A concept expression C is satisfiable with respect to a knowledge base Σ if there exists a model $\mathcal{I}$ of Σ such that $C^{\mathcal{I}}$ is nonempty, $(\exists \mathcal{I}, \mathcal{I} \models \Sigma \wedge \mathcal{I} \models C)$. $\mathcal{I}$ is also a model of C.*

  Note that the Definition 1 resembles the definition of entailment, but not the same. Thus, we also present here the definition of entailment and its complement for the sake of clarity.

  Definition 2 ***(Entailment)*** *A concept C is entailed by a knowledge base Σ ($\Sigma \models C$) if, for every models $\mathcal{I}$ of Σ, each $\mathcal{I}$ is also a model of C $(\forall \mathcal{I}, \mathcal{I} \models \Sigma \rightarrow \mathcal{I} \models C)$.*

  Definition 3 ***(Non-entailment)*** *A concept C is not entailed by a knowledge base Σ ($\Sigma \not\models C$) if there exists model $\mathcal{I}$ of Σ such that $\mathcal{I}$ is not a model of C, i.e., $C^{\mathcal{I}}$ is empty $(\exists \mathcal{I}, \mathcal{I} \models \Sigma \wedge \mathcal{I} \not\models C)$.*

  However, entailment problem can be reduced to satisfiability problems.

  Definition 4 ***(Entailment Reduction to Unsatisfiability)*** *A concept C is entailed by a knowledge base Σ ($\Sigma \models C$) iff $\Sigma \cup \{\neg C\}$ is unsatisfiable, i.e has no model, or $\forall \mathcal{I}, \mathcal{I} \models \Sigma \rightarrow \mathcal{I} \not\models \neg C$.*

  Definition 5 ***(Non-entailment Reduction to Satisfiability)*** *A concept C is not entailed by a knowledge base Σ ($\Sigma \not\models C$) iff $\Sigma \cup \{\neg C\}$ is satisfiable, i.e., model exists, or $\exists \mathcal{I}, \mathcal{I} \models \Sigma \wedge \mathcal{I} \models \neg C$.*

- Subsumption: a concept D subsumes a concept C with respect to $\mathcal{T}$ ($C \sqsubseteq_{\mathcal{T}} D$ or $\mathcal{T} \models C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$.
- Equivalence: two concepts C and D are equivalent with respect to $\mathcal{T}$ ($C \doteq_{\mathcal{T}} D$ or $\mathcal{T} \models C \doteq D$) if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$.
- Disjointness: two concepts C and D are disjoint with respect to $\mathcal{T}$ if, for every model $\mathcal{I}$ of $\mathcal{T}$, $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$.

If the Tbox T is clear from the context, we can drop the qualification "with respect to $\mathcal{T}$ ". In the special case where Tbox is empty, we simply write , $\models C \sqsubseteq D$ if D subsumes C, and , $\models C \doteq D$ if C and D are equivalent.

Subsumption, equivalence and disjointness tests can be reduced to concept (un)satisfiability test, which is, in turn, can be achieved through tableaux algorithm (see section 2.1.3). Consequently, concept satisfiability test is the key inference for Tbox reasoning.

- Subsumption: C is subsumed by D ($C \sqsubseteq D$) iff $C \sqcap \neg D$ is unsatisfiable with respect to $\mathcal{T}$.
- Equivalence: C and D are equivalent ($C \doteq D$) iff both $C \sqcap \neg D$ and $D \sqcap \neg C$ are unsatisfiable with respect to $\mathcal{T}$.
- Disjointness: C and D are disjoint iff $C \sqcap D$ is unsatisfiable with respect to $\mathcal{T}$.

However, in the description logic without full negation, e.g., $\mathcal{AL}$, subsumption and equivalence cannot be reduced to unsatisfiability test.

2.1.2 Reasoning for Abox

Recall that Abox consists of only two kinds of assertion: concept membership assertion of the form $C(a)$ and role membership assertion of the form $R(a,b)$. Hence Abox alone cannot be seen as a knowledge base, it must be coupled with Tbox. Consequently, Abox reasoning will always be done with respect to a Tbox.

In description logic, basic reasoning services for Abox consist of (Baader et al. 2003):

- Instantiation test or instance check: determine whether an assertion is entailed by Abox $\mathcal{A}$ or not. Since, in this work, we consider only $\mathcal{ALC}$ which does not contain any role constructor to form complex roles, role membership assertion test will be easy, i.e., simply find the occurrence of that role assertion in the Abox. At the time of writing, instance check generally refers to only concept membership assertion testing. To check for a concept membership assertion, we just check whether the assertion is entailed by the Abox. The assertion is entailed by Abox ($\mathcal{A} \models C(a)$) if every interpretation that satisfies $\mathcal{A}$, i.e., every model of $\mathcal{A}$, also satisfies $C(a)$.
- Realisation: given an individual a and a set of concepts, find the most specific concept C from the set such that $\mathcal{A} \models C(a)$.
- Retrieval: given an Abox $\mathcal{A}$ and concept C, find all individuals a such that $\mathcal{A} \models C(a)$.
- Abox consistency: Abox $\mathcal{A}$ is consistent iff it is consistent with respect to Tbox $\mathcal{T}$. Consequently, we must use Tbox in this reasoning, i.e., for $\mathcal{ALC}$, expanding the Abox with unfolded Tbox concepts (Tessaris 2001). Unfolded concept or expanded concept C' is obtained by replacing names in the description of the original concept C with their descriptions in $\mathcal{T}$. Note that C is satisfiable with respect to $\mathcal{T}$ iff C' is satisfiable, i.e., the original concept and the consistency preserving expanded/unfolded concept are in fact equivalent, $C \doteq_{\mathcal{T}} C'$ (Horrocks 1997). Therefore expansion of $\mathcal{A}$ with respect to $\mathcal{T}$ (the Abox $\mathcal{A}'$) can be obtained by replacing each concept assertion $C(a)$ in $\mathcal{A}$ with the assertion $C'(a)$. In every model of $\mathcal{T}$, a concept C and its expansion C' are interpreted in the same way. Hence, $\mathcal{A}$ is consistent with respect to $\mathcal{T}$ iff $\mathcal{A}'$ is consistent. $\mathcal{A}'$ is consistent iff it is satisfiable, i.e., there is at least a nonempty model for $\mathcal{A}'$. Note that $\mathcal{A}'$ also represents the whole knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$.

Theorem 6 ***(Expanded Abox)*** *Given a satisfiable Tbox $\mathcal{T}$, an Abox $\mathcal{A}$ is satisfiable with respect to $\mathcal{T}$ iff its expansion $\mathcal{A}'$ is satisfiable.*

Proof *see (Baader et al. 2003)*

It is easy to see that Theorem 7 logically follows from Theorem 6.

Theorem 7 ***(Expanded Abox Entailment)*** *Given a satisfiable Tbox $\mathcal{T}$, a knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ entails a concept expression Q ($\langle \mathcal{T}, \mathcal{A} \rangle \models Q$) iff $\mathcal{A}'$ entails Q ($\mathcal{A}' \models Q$).*

Realisation and retrieval can be reduced to instantiation test. They can be done through series of instantiation tests. In addition, we can reduce instantiation test to consistency problem for Abox since $\mathcal{A} \models C(a)$ iff $\mathcal{A} \cup \{\neg C(a)\}$ is inconsistent. Concept satisfiability can also be reduced to Abox consistency test since C is satisfiable iff $\{C(a)\}$ is consistent, where a is an arbitrary individual name.

At this point, it turns out that there is one main inference problem, i.e., consistency check for Abox, to which all other reasoning services can be reduced. Since tableaux algorithms can be used to solve such test, thus all of the reasoning services in description logic can be achieved through the application of tableaux algorithm (Baader et al. 2003). In addition, most of the practical description logic systems, such as FACT (Horrocks 1997) and RACER (Haarslev & Möller 2002), exploit tableaux algorithm as the basis for their reasoning. In fact, tableaux becomes the de-facto standard for reasoning in description logic. We will explain tableaux in section 2.1.3.

2.1.3 Tableaux algorithm

Traditionally, tableaux algorithm was designed to prove the satisfiability of a concept expression. The main idea behind this algorithm (Baader & Hollunder 1991) is based on notational variant of the first order tableaux calculus. In fact, a tableaux algorithm tries to prove the satisfiability of a concept expression C by demonstrating a nonempty model of C. It constructively builds a model for a given concept. If it can build a model, then the concept is satisfiable.

First, we will present tableaux algorithm for testing the satisfiability of a Tbox concept and describe its operation. Then, we will show how this algorithm can also be applied to Abox consistency test. However, this topic is intended to illustrate the general principles only. For proof of termination, soundness and completeness of the algorithm, consult (Baader et al. 2003).

Before we proceed with the algorithm, we need to introduce an appropriate data structure for representation of each tableau. The original paper by Schmidt-Schauβ and Smolka (Schmidt-Schauß & Smolka 1991), and also many other papers on tableaux algorithms for description logic (Baader et al. 2003), exploit a notion of a constraint system for this purpose. A constraint system is a set of constraints, which are syntactic elements of the forms:

- $x : C$ concept constraint
- $\langle x, y \rangle : R$ role constraint
- $x \neq y$ inequality constraint

Although the syntax is similar to the one for Abox assertions, there is a difference in the meaning of the x and y terms. Since we are going to test the satisfiability of a Tbox concept expression, no individual is involved. Consequently, x and y are not individuals, but variables, so the unique name assumption does not apply to them unless stated explicitly by an inequality constraint. The presence of at-most number restriction may lead to the identification of different individual names. However, our considered description logic does not contain at-most number restriction constructor, thus we do not need inequality constraint here. A constraint system can be seen as a graph where the nodes are individual variable names, and the edges corresponds to role names. Each node has associated concept expressions which are, in fact, constraints of the variable corresponded to the node.

To test satisfiability of a concept expression C with respect to Tbox $\mathcal{T}$, firstly, unfold the concept expressions with their definitions defined in $\mathcal{T}$, and get the new expanded concept C'. Next, transform the concept expression into a negation normal form (NNF), i.e., push negation signs as far as possible into the description, using combination of de Morgan's law and usual rules for quantifiers, for example $\neg\exists R.C = \forall R.\neg C$ and $\neg\forall R.C = \exists R.\neg C$. The concept expression is in negation normal form if negations apply only to concept names, not to any compound terms. For example the concept $C = \neg\exists R.A \sqcap \neg(\exists R.D \sqcup \forall S.\neg D)$ can be transformed into NNF, $C_0 = \forall R.\neg A \sqcap \forall R.\neg D \sqcap \exists S.D$. Now, we get a new concept expression as a constraint system S.

The process of constructing a model proceeds by completing (or extending) a constraint system S, using a set of

consistency preserving transformation-completion (or expansion) rules in figure 2.1.3. These rules modify a constraint system by adding or rewriting the constraints. Apply these rules to the constraint system until no more rules apply. When no rule is applicable, the constraint system is said to be *completed* if there is no obvious contradiction, so called *clash*, i.e., $\{x:A, x:\neg A\}$ or $\{x:\bot\}$, occurs. The concept C' is consistent iff there exists a completed constraint system, i.e., a nonempty model. Since $C \doteq_{\mathcal{T}} C'$, the concept C is consistent with respect to $\mathcal{T}$ iff the expanded concept C' is consistent.

The $\rightarrow_{\sqcap}$ -rule
Condition: $x : C_1 \sqcap C_2$ is in S, and both $x : C_1$ and $x : C_2$ are not in S.
Action: $S = S \cup \{x : C_1, x : C_2\}$

The $\rightarrow_{\sqcup}$ -rule
Condition: $x : C_1 \sqcup C_2$ is in S, and neither $x : C_1$ nor $x : C_2$ is in S.
Action: $S' = S \cup \{x : C_1\}$, $S'' = S \cup \{x : C_2\}$

The $\rightarrow_{\forall}$ -rule
Condition: $x : \forall R.C$ is in S, $\langle x, y\rangle : R$ is in S, and $y : C$ is not in S.
Action: $S = S \cup \{y : C\}$

The $\rightarrow_{\exists}$ -rule
Condition: $x : \exists R.C$ is in S,
and there is no z such that both $\langle x, z\rangle : R$ and $z : C$ are in S.
Action: $S = S \cup \{\langle x, y\rangle : R, y : C\}$ where y is new variable name.

Figure 1: Completion rules for $\mathcal{ALC}$ satisfiability test

Note that the only rule that is non-deterministic is the disjunction rule ($\rightarrow_{\sqcup}$). For more detail proof of these rules, see (Donini, Lenzerini, Nardi & Schaerf 1996).

In order to test Abox consistency using tableaux algorithm, we simply allow x and y in constraint system notation to be able to represent individual name. Consequently, concept constraint and role constraint in a constraint system become concept membership assertion and role membership assertion in Abox respectively. The algorithm is the same as above, except, instead of a concept expression, we try to prove satisfiability of the whole expanded Abox $\mathcal{A}'$.

2.2 Reasoning Complexity and Optimization

Since, all basic reasoning services can be reduced to satisfiability problem, which, in turn, can be achieved through tableaux algorithm, complexity of reasoning services provided in almost description logic systems are based on complexity of tableaux algorithm. Recall that description logic was invented because of inefficiency of first-order logic. In fact, description logic is one of two major approaches for mitigation of inefficiency of reasoning for logic. Its prominent significance is that trade-offs between expressive power and efficiency of reasoning were studied throughout a decade.

We denote problem that is solvable using algorithm with polynomial time (PTIME) worst-case complexity as *tractable* problem, and *intractable* problem otherwise. Normally, tractable is preferred. However, there is worse class of problem, *undecidable* problem. We says the problem is undecidable if there is no solving algorithm that terminates, and *decidable* otherwise. To the best of our knowledge, the most expressive description logic that its concept satisfiability problem is decidable in PTIME worst-case complexity is $\mathcal{ALN}$.

Logics with PTIME worst-case complexity have been criticized for their too limited expressive power (Doyle & Patil 1991). Thus we need the language that is more expressive. Consequently, we choose $\mathcal{ALC}$ which is expressive enough, since it is a subset of mostly every expressive description logics (Donini & Massacci 2000).

3 The Efficiency Issue

In DL, there are two standard types of queries allowed, i.e., boolean query and non-boolean query, which are in turn instance checking (or instantiation test) and retrieval Abox reasoning services respectively.

A boolean query $\mathcal{Q}_b$ refers to a formula of the form

$$\mathcal{Q}_b \leftarrow QExp,$$

where $QExp$ is an assertion about individual, e.g.,

$$\mathcal{Q}_b \leftarrow Tom : (Parent \sqcap \exists hasChild.Employee)$$

The query will return one of the member of the boolean set $\{True, False\}$. $\mathcal{Q}_b$ will return $True$ if and only if every interpretation that satisfies the knowledge base $\mathcal{K}$ also satisfies $QExp$, and return $False$ otherwise.

A non-boolean query $\mathcal{Q}_{nb}$ refers to a formula of the form

$$\mathcal{Q}_{nb} \leftarrow QExp,$$

where $QExp$ is a concept expression, e.g.

$$\mathcal{Q}_{nb} \leftarrow Parent \sqcap \exists hasChild.Employee$$

In this case the query will return one of the member of the set $\{\bot, \mathcal{M}\}$, where $\bot$ refers to the empty set, and $\mathcal{M}$ represents a set of models $\{\mathcal{M}_1, \ldots, \mathcal{M}_m\}$, where each of them satisfies $QExp$ with respect to the knowledge base $\mathcal{K}$. The query will return $\mathcal{M}$ if and only if there exists at least one such model, otherwise return $\bot$.

A non-boolean query (retrieval) can be trivially transformed into a set of boolean queries for all candidate tuples, i.e., retrieving sets of tuples can be achieved by repeated application of boolean queries with different tuples of individuals substituted for variables. However, answering a boolean query is in fact an entailment problem. For example, answering the boolean query:

$$\mathcal{Q}_b \leftarrow Tom : (Parent \sqcap \exists hasChild.Employee),$$

is the problem of checking whether

$$\mathcal{K} \models Tom : (Parent \sqcap \exists hasChild.Employee).$$

In a DL (supporting full negation, e.g., $\mathcal{ALC}$), boolean query or instance checking can be reduced to knowledge base satisfiability test: $\mathcal{K} \models C(a)$ iff $\mathcal{K} \cup \{\neg C(a)\}$ is unsatisfiable.

(Donini & Massacci 2000) gave a tableaux algorithm for solving $\mathcal{ALC}$ satisfiability problem with respect to a Tbox. They proved that their algorithm has EXPTIME-complete worst-case complexity. To the best of our knowledge, this is the latest known result of complexity proof for the $\mathcal{ALC}$ satisfiability problem with respect to a Tbox. Nevertheless, the ontology language OWL, in particular OWL-DL, of the Semantic web technology is based on $\mathcal{SHOIQ}(\mathcal{D})$ which is even more expressive than $\mathcal{ALC}$. Since the query answering is in fact an instance checking (or a retrieval reasoning service) which can be reduced to a satisfiability problem. It is easy to verify that the existing DL reasoning services are still not enough to be used solely with the Semantic web technology. One way to mitigate the problem is to optimize the algorithm more and more. We propose an optimization technique for answering a query over a description logic knowledge base. This technique is coherent with nature of the Web in that it supports multiple-Abox environment, which is corresponding to multiple data source environment in the Web.

4 The Approach

This contribution focuses on finding an answer to the question: "How can we (efficiently) answer a query in $\mathcal{ALC}$ based-Semantic Web system, given single ontology $\mathcal{T}$, and multiple data sources $\mathcal{A}$s, examining the minimum number of data sources?". We refer to an Abox as a data source.

The idea of this section is based on the observation that $2^m > 2^n + 2^p$, where $m = n + p$ for $n, p > 1$. This means that if we can split the search space into independent parts, query the parts independently from each other, and combine the answers, then we have a considerable improvement of the performance of the query system. This idea agrees with the understanding of the Semantic Web as a collection of sometime "unrelated" data sources. In addition we propose to attach to each data source a data source description (or source description), a compact representation of the content of the data page. This idea is similar to the intuition behind indexes in databases. In the same way that type of indexes is more or less appropriate for particular queries, source descriptions depend on the type of queries. On the other hand, as we will see in the rest of this section, the relationships among data sources are not influenced by queries. They are determined by the structure of the data itself.

In this approach, the knowledge base architecture is shown in the following figure:

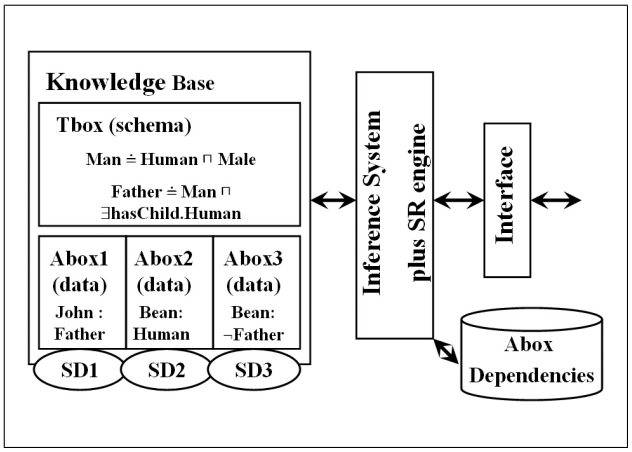

Figure 2: The Knowledge Base Architecture

The intuition here is to associate to every Abox $\mathcal{A}$ a source description $SD(\mathcal{A})$, and to supplement the inference engine with information about the mutual dependencies of the Aboxes in the system, in order to determine which Aboxes are relevant and must be queried.

The first step is to associate to every Abox its domain.

Definition 8 *Given an Abox $\mathcal{A}$, let $H_{\mathcal{A}}$ be the Herbrand universe of $\mathcal{A}$ (i.e., the set of all the individual occurring in expression in $\mathcal{A}$). For any interpretation $\mathcal{I}$, $\Delta^{\mathcal{I}}_{\mathcal{A}}$, the domain of $\mathcal{A}$ is defined as follows:*

$$\Delta^{\mathcal{I}}_{\mathcal{A}} = \left\{ d \in \Delta^{\mathcal{I}} | a \in H_{\mathcal{A}} \wedge a^{\mathcal{I}} = d \right\}.$$

Definition 9 (Multiple Assertional Knowledge Base) *Given a set $\bar{\mathcal{A}}$ of Aboxes $\mathcal{A}_1, \ldots, \mathcal{A}_k$, i.e., $\bar{\mathcal{A}} = \{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$ and a Tbox $\mathcal{T}$, the* multiple assertional knowledge base *is the knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{A}$ is the Abox obtained from the union of all the Aboxes in $\bar{\mathcal{A}}$, i.e., $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \ldots \cup \mathcal{A}_k$.*

A consequence of the above definition is that the interpretation domain of $\mathcal{A}$ is equivalent to the union of interpretation domains of the $\mathcal{A}_j$s ($\Delta^{\mathcal{I}}_{\mathcal{A}} = \bigcup_{1 \le j \le k} \Delta^{\mathcal{I}}_{\mathcal{A}_j}$). Since $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ by definition, thus, for arbitrary C, $C^{\mathcal{I}}_{\mathcal{A}} = \bigcup C^{\mathcal{I}}_{\mathcal{A}_j}$, for $j \in \{1, \ldots, k\}$ and $C_{\mathcal{A}_j}$ is the concept C that occurs in a concept membership assertion in $\mathcal{A}_j$.

We approach the problem in 5 steps:

1. Determine dependencies among data sources, and group data sources which are dependent on each other together.
2. Associate each data source (or group of data sources) with a source description.
3. When one queries the knowledge base, exploit a procedure to find irrelevant data sources (or groups of data sources) with respect to the query, taking into account source descriptions and the query. Eliminate the irrelevant data sources (or groups of data sources) from query answering process, yielding a set of possible relevant data sources (or groups of data sources) to be queried.
4. For each remaining data source (or group of data sources) from the previous step, apply the existing query answering procedure to each of them, yielding answer from each of them.
5. Simply combine answers from the queried data sources (or groups of data sources) together, since each data source (or group of data sources) is independent with the other.

Since a reasoning procedure for simple query answering in the fourth step exists (Tessaris 2001), we will focus on other steps, which are in fact the steps of the data source space partitioning and reduction using source description.

The approach can be implemented by the following algorithm.

Algorithm 1 *partitioned_QA*$(query, \mathcal{A})$:

```
depset = {}
answer = False
for all A_m ∈ Ā do
    AG(A_m) = create_abox_graph(A_m)
end for
for all 2-combinations {A_i, A_j} of A do
    if find_abox_dependency(AG(A_i),AG(A_j)) = True
        then add dep(A_i, A_j) to depset
    end if
end for
A^g = combine_dependent_abox(A,depset)
create_source_description(A^g)
for all A_h ∈ A^g do
    if query_relevancy(SD(A_h),query) = True then
        answer = answer ∨ instance_checking(A_h,query)
    end if
end for
```

First, since an Abox $\mathcal{A}_i$ can overlap with another Abox $\mathcal{A}_j$, we must consider multiple Aboxes at the same time. However, we will not treat all of the Aboxes as a single Abox, because, in this case, the associated reasoning is computational expensive. Consequently, we need some additional procedure to determine dependencies among Aboxes since we need to know which Aboxes should be considered together. In other word, we need to group dependent Aboxes together and treat them as a new single Abox consisting of multiple dependent Aboxes. To make this clear, we need to formally define the dependency between Aboxes in the context of Abox reasoning.

Firstly, we will introduce graph notation for an Abox.

Definition 10 (Abox Graph) *An Abox graph for an Abox $\mathcal{A}$, $AG(\mathcal{A})$, consists of a set N of nodes (vertexes), a set E of edges (arcs), and a function f from E to $\{(a,b) \mid a,b \in N\}$. Each edge, label ed R_i, represents exactly a role name of a role membership assertion $R_i(a,b) \in \mathcal{A}$. Hence, each*

node represents exactly one individual name. An Abox graph is a directed multigraph.

The create_abox_graph function will produce an Abox graph $AG(\mathcal{A}_m)$ for each Abox $\mathcal{A}_m$. We will say that an Abox $\mathcal{A}$ is *connected* if its Abox graph $AG(\mathcal{A})$ is weakly connected.

Definition 11 (Abox Dependency) *Given two connected Aboxes $\mathcal{A}_1$ and $\mathcal{A}_2$, where $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$; $\mathcal{A}_1$ and $\mathcal{A}_2$ depend on each other if the graph of Abox $\mathcal{A}$ is (weakly) connected, and independent otherwise.*

Proposition 12 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two independent Aboxes in multiple assertional knowledge base. Let $\Delta^{\mathcal{I}}_{\mathcal{A}_1}$ and $\Delta^{\mathcal{I}}_{\mathcal{A}_2}$ be the domains of $\mathcal{A}_1$ and $\mathcal{A}_2$, then:*

- $\Delta^{\mathcal{I}}_{\mathcal{A}_1} \cap \Delta^{\mathcal{I}}_{\mathcal{A}_2} = \emptyset$;
- *for any concept C, $C^{\mathcal{I}}_{\mathcal{A}_1} \cap C^{\mathcal{I}}_{\mathcal{A}_2} = \emptyset$, where $C^{\mathcal{I}}_{\mathcal{A}_i}$ is the extension of C in $\Delta^{\mathcal{I}}_{\mathcal{A}_i}$.*

If $\mathcal{A}$ is unconnected, i.e., $\mathcal{A}_1$ and $\mathcal{A}_2$ are independent on each other, then it means that $\mathcal{A}_1$ and $\mathcal{A}_2$ do not share any common node (individual) because Aboxes $\mathcal{A}_1$ and $\mathcal{A}_2$ are already connected by themselves. Thus, we can use Abox graphs to determine Abox dependency.

For any unordered pair of Aboxes $\{\mathcal{A}_i, \mathcal{A}_j\}$, we determine the Abox dependency between the two Aboxes ($\mathcal{A}_i$ and $\mathcal{A}_j$). According to the definition, Abox dependency can be detected using the connectivity of the Abox graph of $\mathcal{A}$, i.e., $AG(\mathcal{A})$, where $\mathcal{A} = \mathcal{A}_i \cup \mathcal{A}_j$. Thus, we can exploit any UCONN (undirected graph connectivity problem) algorithm for this purpose. The function find_abox_dependency($AG(\mathcal{A}_i)$,$AG(\mathcal{A}_j)$) returns True if two Aboxes $\mathcal{A}_i$ and $\mathcal{A}_j$ depend on each other, and False otherwise. If the function returns True, then we add dep($\mathcal{A}_i, \mathcal{A}_j$) to the set "depset", i.e., the set that stores dependency value of each pair of Aboxes. Then we virtually combine dependent Aboxes together as a group by the function combine_dependent_abox($\mathcal{A}$,depset). The Abox $\bar{\mathcal{A}}$ will become $\mathcal{A}^g$, i.e., the set of already-grouped Aboxes and ungrouped Aboxes. Each Abox in $\mathcal{A}^g$ is independent of each other.

Next, we need to show two things:

1. if two Aboxes depend on each other, then a DL reasoning service, in particular instance checking and retrieval, needs to take into account the two Aboxes together;
2. if two Aboxes are independent of each other, then a DL reasoning over the two Aboxes can be done separately over each of them.

The following theorem supports the last step in our approach. It provides the reason why we can simply combine the answer from each $\mathcal{A}_i \in \mathcal{A}^g$ together. In other words it states that the the instance checking (a query answering) problem over $\mathcal{A}^g$ can be reduced to separate instance checking problems over each $\mathcal{A}_i$.

Theorem 13 (Independent Abox for Boolean Query) *Given two connected Aboxes $\mathcal{A}_1$ and $\mathcal{A}_2$, where $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$, If $\mathcal{A}_1$ and $\mathcal{A}_2$ are independent on each other, then for any boolean query $\mathcal{Q}$ and Tbox $\mathcal{T}$, $\langle \mathcal{T}, \mathcal{A} \rangle \models \mathcal{Q}$ if and only if $\langle \mathcal{T}, \mathcal{A}_1 \rangle \models \mathcal{Q}$ or $\langle \mathcal{T}, \mathcal{A}_2 \rangle \models \mathcal{Q}$.*

Proof First, we prove the only if direction, and we will assume that both $\mathcal{A}_1$ and $\mathcal{A}_2$ are consistent with $\mathcal{K}$, since if one of them is not then the theorem trivially holds.

Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are *independent* on each other, by Proposition 12, we have $\Delta^{\mathcal{I}}_1 \cap \Delta^{\mathcal{I}}_2 = \emptyset$, where $\Delta^{\mathcal{I}}_1$ and $\Delta^{\mathcal{I}}_2$ are the domains of $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively.

Suppose $\langle \mathcal{T}, \mathcal{A}_1 \rangle \not\models \mathcal{Q}$ and $\langle \mathcal{T}, \mathcal{A}_2 \rangle \not\models \mathcal{Q}$. These mean $\exists \mathcal{I}_1$ such that $\mathcal{I}_1 \models \mathcal{A}_1$, $\mathcal{I}_1 \models \mathcal{T}$, $\mathcal{I}_1 \models \neg\mathcal{Q}$, and $\exists \mathcal{I}_2$ such that $\mathcal{I}_2 \models \mathcal{A}_2$, $\mathcal{I}_2 \models \mathcal{T}$ and $\mathcal{I}_2 \models \neg\mathcal{Q}$. Note that $\mathcal{I}_1$ and $\mathcal{I}_2$ are arbitrary interpretations of $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively with the only constraint of being interpretations of $\mathcal{T}$.

Since $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ and $\Delta^{\mathcal{I}}_1 \cap \Delta^{\mathcal{I}}_2 = \emptyset$, we can create an interpretation $\mathcal{I}$ of $\mathcal{A}$ such that $\mathcal{I}$ is the union of the interpretation $\mathcal{I}_1$ of $\mathcal{A}_1$ and the interpretation $\mathcal{I}_2$ of $\mathcal{A}_2$ ($\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2$). More precisely, $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \bullet^{\mathcal{I}} \rangle$ is defined as follows:

(i) $\Delta^{\mathcal{I}} = \Delta^{\mathcal{I}}_1 \cup \Delta^{\mathcal{I}}_2$ because $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$, where $\Delta^{\mathcal{I}}$ is the domains of $\bar{\mathcal{A}}$

(ii) For any constant a,

$$a^{\mathcal{I}} = \begin{cases} a^{\mathcal{I}_1} & \text{if } a \text{ occurs in } \mathcal{A}_1 \\ a^{\mathcal{I}_2} & \text{if } a \text{ occurs in } \mathcal{A}_2 \end{cases}$$

(iii) For any concept C, $C^{\mathcal{I}} = C^{\mathcal{I}_1} \cup C^{\mathcal{I}_2}$

(iv) For any role R, $R^{\mathcal{I}} = R^{\mathcal{I}_1} \cup R^{\mathcal{I}_2}$

Since $\Delta^{\mathcal{I}}_1 \cap \Delta^{\mathcal{I}}_2 = \emptyset$, then it is immediate to verify that $\mathcal{I}$ is indeed an interpretation, and $\mathcal{I} \models \mathcal{T}$, since $\mathcal{I}_1 \models \mathcal{T}$ and $\mathcal{I}_2 \models \mathcal{T}$.

Since $\mathcal{I}_1 \models \neg\mathcal{Q}$ and $\mathcal{I}_2 \models \neg\mathcal{Q}$, from (iii), we can immediately verify $\mathcal{I} \models \neg\mathcal{Q}$, i.e., $(\neg\mathcal{Q})^{\mathcal{I}} = (\neg\mathcal{Q})^{\mathcal{I}_1} \cup (\neg\mathcal{Q})^{\mathcal{I}_2}$, where $\mathcal{I}$ is the interpretation of $\mathcal{A}$. From (ii), (iii) and (iv), we can also infer that $(\mathcal{A})^{\mathcal{I}} = (\mathcal{A}_1)^{\mathcal{I}_1} \cup (\mathcal{A}_2)^{\mathcal{I}_2}$, i.e., $\mathcal{I} \models \mathcal{A}$.

Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are assumed to be consistent by themselves, we only need to prove that there is no clash between $\mathcal{A}_1$ and $\mathcal{A}_2$. For an arbitrary concept C, by general definition in description logic, we get $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$. In addition, we get $(\neg C)^{\mathcal{I}} = (\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}) \subseteq \Delta^{\mathcal{I}}$. Thus, for arbitrary C, $C^{\mathcal{I}_1} \subseteq \Delta^{\mathcal{I}}_1$ and $(\neg C)^{\mathcal{I}_2} \subseteq \Delta^{\mathcal{I}}_2$. Since $\Delta^{\mathcal{I}}_1 \cap \Delta^{\mathcal{I}}_2 = \emptyset$, therefore, $C^{\mathcal{I}_1} \cap (\neg C)^{\mathcal{I}_2} = \emptyset$, which infers that no clash can occur between $\mathcal{A}_1$ and $\mathcal{A}_2$.

Thus for the interpretation $\mathcal{I}$ of $\mathcal{A}$, we have $(\mathcal{A})^{\mathcal{I}} \neq \emptyset$ and $(\neg\mathcal{Q})^{\mathcal{I}} \neq \emptyset$, i.e., $\mathcal{I} \models \mathcal{A} \wedge \mathcal{I} \models \neg\mathcal{Q}$ which is the definition of $\mathcal{A} \not\models \mathcal{Q}$. Therefore, $\mathcal{A} \models \mathcal{Q}$ only if $\mathcal{A}_1 \models \mathcal{Q}$ or $\mathcal{A}_2 \models \mathcal{Q}$ which infers $\langle \mathcal{T}, \mathcal{A} \rangle \models \mathcal{Q}$ only if $\langle \mathcal{T}, \mathcal{A}_1 \rangle \models \mathcal{Q}$ or $\langle \mathcal{T}, \mathcal{A}_2 \rangle \models \mathcal{Q}$.

For the if direction, we assume that either 1) $\langle \mathcal{T}, \mathcal{A}_1 \rangle \models \mathcal{Q}$ or 2) $\langle \mathcal{T}, \mathcal{A}_2 \rangle \models \mathcal{Q}$. In both cases, by monotonicity, we obtain $\langle \mathcal{T}, \mathcal{A}_1 \cup \mathcal{A}_2 \rangle \models \mathcal{Q}$ which is $\langle \mathcal{T}, \mathcal{A} \rangle \models \mathcal{Q}$. □

In the second step of the approach, we associate each Abox (or group of Aboxes) with a source description, using create_source_description($\mathcal{A}^g$). A source description can be view as a surrogate of each data source. Surrogate refers to a brief representation of an information source that is designed to convey an indication of the information source's intent (Goodchild 1998). A good surrogate has two major properties: (1) it corresponds to some common understanding in the user's community, and (2) it can be organized in a way that is searchable.

Source descriptions are used to determine the relevancy of each Abox $\mathcal{A}_h \in \mathcal{A}^g$ with respect to a query. Source descriptions depend on the type of the query. For boolean queries, the source description of each Abox $\mathcal{A}_h \in \mathcal{A}^g$ can be a simple list of all individuals appearing in the Abox $\mathcal{A}_h$. The idea is if the query does not satisfy $SD(\mathcal{A}_h)$ (necessary and sufficient conditions), it is guaranteed that the query over Abox $\mathcal{A}_h$ will fail, i.e., it returns False. This is done by the function query_relevancy($SD(\mathcal{A}_h)$,*query*). This function returns False if the query does not satisfy $SD(\mathcal{A}_h)$, i.e., the Abox $\mathcal{A}_h$ is fully irrelevant to the query, and will contribute nothing to the answer of the query. The function works by extracting an individual from the query, then checking if it is in the source description $SD(\mathcal{A}_h)$ or not. If it is, then it queries the Abox

$\mathcal{A}_h$, using normal boolean query answering procedure instance_checking($\mathcal{A}_h$,*query*).

This can be formalised as follows:

Definition 14 *Let $\mathcal{A}$ be an Abox, the boolean query source description for $\mathcal{A}$ ($SD_b(\mathcal{A})$) is the the Herbrand universe of $\mathcal{A}$, i.e., $SD_b(\mathcal{A}) = H_{\mathcal{A}}$.*

We can now prove soundness and completeness of the proposed algorithm.

Theorem 15 (Soundness and Completeness) *Let $\mathcal{Q}$ be a boolean query. Let $\mathcal{A} \not\sharp \mathcal{Q}$ represents when query_relevance($SD_b(\mathcal{A}),\mathcal{Q}$) returns False, i.e., $\mathcal{A}$ is not relevant the query $\mathcal{Q}$, and let $\mathcal{A}\sharp\mathcal{Q}$ represents otherwise. If $\mathcal{A} \not\sharp \mathcal{Q}$ then $\langle \mathcal{T}, \bar{\mathcal{A}} - \{\mathcal{A}\}\rangle \models \mathcal{Q}$ if and only if $\langle \mathcal{T}, \bar{\mathcal{A}}\rangle \models \mathcal{Q}$.*

Ah,not just A

Proof Suppose $\mathcal{A} \not\sharp \mathcal{Q}$. This means $a \notin SD_B(\mathcal{A})$, where $\mathcal{Q}$ is $C(a)$.

First, we prove the only if direction. Suppose $\langle \mathcal{T}, \bar{\mathcal{A}} - \{\mathcal{A}\}\rangle \models \mathcal{Q}$. However, $\bar{\mathcal{A}} - \{\mathcal{A}\} \subseteq \bar{\mathcal{A}}$. By monotonicity, we obtain $\langle \mathcal{T}, \bar{\mathcal{A}}\rangle \models \mathcal{Q}$.

Therefore, $\langle \mathcal{T}, \bar{\mathcal{A}} - \{\mathcal{A}\}\rangle \models \mathcal{Q}$ only if $\langle \mathcal{T}, \bar{\mathcal{A}}\rangle \models \mathcal{Q}$.

For the if direction, suppose $\langle \mathcal{T}, \bar{\mathcal{A}}\rangle \models \mathcal{Q}$. We, then, prove by case.

Case 1: $\mathcal{Q}$ is a tautology. It is immediate to verify that $\langle \mathcal{T}, \bar{\mathcal{A}} - \{\mathcal{A}\}\rangle \models \mathcal{Q}$ is true.

Case 2: $\mathcal{Q}$ is not a tautology. From Lemma 16, we obtain $\langle \mathcal{T}, \mathcal{A}\rangle \not\models \mathcal{Q}$. In addition, $\langle \mathcal{T}, \bar{\mathcal{A}}\rangle \models \mathcal{Q}$ is equal to $\langle \mathcal{T}, \bar{\mathcal{A}} - \{\mathcal{A}\} \cup \{\mathcal{A}\}\rangle \models \mathcal{Q}$. By Theorem 13, we get $\langle \mathcal{T}, \bar{\mathcal{A}} - \{\mathcal{A}\}\rangle \models \mathcal{Q}$ or $\langle \mathcal{T}, \mathcal{A}\rangle \models \mathcal{Q}$. Since $\langle \mathcal{T}, \mathcal{A}\rangle \not\models \mathcal{Q}$, we obtain $\langle \mathcal{T}, \bar{\mathcal{A}} - \{\mathcal{A}\}\rangle \models \mathcal{Q}$.

These cases cover all possibilities. Therefore, $\langle \mathcal{T}, \bar{\mathcal{A}} - \{\mathcal{A}\}\rangle \models \mathcal{Q}$ if $\langle \mathcal{T}, \bar{\mathcal{A}}\rangle \models \mathcal{Q}$.

Therefore, if $\mathcal{A} \not\sharp \mathcal{Q}$ then $\langle \mathcal{T}, \bar{\mathcal{A}} - \{\mathcal{A}\}\rangle \models \mathcal{Q}$ if and only if $\langle \mathcal{T}, \bar{\mathcal{A}}\rangle \models \mathcal{Q}$. □

Lemma 16 *Let $\mathcal{Q}$ be a boolean query $C(a)$. If $a \notin SD_B(\mathcal{A})$ and $\mathcal{Q}$ is not a tautology, then $\langle \mathcal{T}, \mathcal{A}\rangle \not\models \mathcal{Q}$.*

Proof Suppose $a \notin SD_B(\mathcal{A})$. By definition, this means $a \notin \Delta^{\mathcal{I}}_{\mathcal{A}}$. In other words, there is no a in $\mathcal{A}$, i.e., $C(a)$ is definitely not in $\mathcal{A}$. Supppose $\mathcal{Q}$ is not a tautology. At this state, we want to prove $\langle \mathcal{T}, \mathcal{A}\rangle \not\models C(a)$ which is equal to proving that $\langle \mathcal{T}, \mathcal{A}\rangle \cup \{\neg C(a)\}$ is satisfiable. Since $\langle \mathcal{T}, \mathcal{A}\rangle$ is consistent, $\langle \mathcal{T}, \mathcal{A}\rangle \cup \{\neg C(a)\}$ will be unsatisfiable only if either $C(a)$ is in $\mathcal{A}$ or $C(a)$ is a tautology. Since neither of them is true, $\langle \mathcal{T}, \mathcal{A}\rangle \cup \{\neg C(a)\}$ is satisfiable. Consequently, we prove $\langle \mathcal{T}, \mathcal{A}\rangle \not\models C(a)$.

Therefore, if $a \notin SD_B(\mathcal{A})$ and $\mathcal{Q}$ is not a tautology, then $\langle \mathcal{T}, \mathcal{A}\rangle \not\models \mathcal{Q}$. □

Finally, in the last step we simply combine the answers together using disjunction. Again this step is justified by Theorem 13.

5 Example for the Approach

Suppose we have a knowledge bases $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$, where the data is distributed over four Aboxes, i.e., $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3 \cup \mathcal{A}_4$.

Suppose the Tbox $\mathcal{T}$ is as follows:

$$\begin{aligned}\mathcal{T} = \{&Male \doteq \neg Female,\\ &Man \doteq Human \sqcap Male,\\ &Woman \doteq Human \sqcap Female,\\ &Father \doteq Man \sqcap \exists hasChild.Human,\\ &Mother \doteq Woman \sqcap \exists hasChild.Human,\\ &Parent \doteq Father \sqcup Mother,\\ &Employee \doteq Human \sqcap \exists workFor.Org,\\ &Org \doteq ProfitableOrg \sqcup CharityOrg,\\ &ProfitableOrg \doteq \neg CharityOrg,\\ &ProfitableOrg \doteq Company \sqcup Partnership \sqcup SoleProprietorship\\ &NewspaperCompany \doteq Company \sqcap \exists publish.Newspaper\}\end{aligned}$$

Suppose that the four Aboxes are as follows:

$$\begin{aligned}\mathcal{A}_1 = \{&Man(John), Man(Clark), Woman(Chloe),\\ &SoleProprietorship(BKKDelight), hasChild(John, Clark),\\ &workFor(Clark, DailyPlanet), attend(Clark, MetroNews05),\\ &hasChild(John, Chloe), workFor(Chloe, BKKDelight),\\ &enrolAt(Chloe, DKE)\}\\ \mathcal{A}_2 = \{&ResearchGroup(DKE), School(ENG), University(UQ),\\ &Workshop(MetroNews05), partOf(DKE, ENG),\\ &facultyOf(ENG, UQ), locatedIn(UQ, Australia)\}\\ \mathcal{A}_3 = \{&NewspaperCompany(DailyPlanet), Workshop(MetroNews05),\\ &NewspaperCompany(Inquisitor), locatedIn(DailyPlanet, USA),\\ &sponsoredBy(MetroNews05, DailyPlanet),\\ &sponsoredBy(MetroNews05, Inquisitor),\\ &sponsoredBy(MetroNews05, DKE)\}\\ \mathcal{A}_4 = \{&CharityOrg(WorldHelp), CharityProgram(TsuHelp),\\ &CharityProgram(QuakeHelp),\\ &locatedIn(WorldHelp, Germany),\\ &create(WorldHelp, TsuHelp), create(WorldHelp, QuakeHelp)\}\end{aligned}$$

We simply create the Abox graph for each Abox, yielding four Abox graphs: $AG(\mathcal{A}_1)$, $AG(\mathcal{A}_2)$, $AG(\mathcal{A}_3)$, and $AG(\mathcal{A}_4)$. For every combination of two Aboxes of $\mathcal{A}$, we determine the Abox dependency between them using the find_abox_dependency function. The function will combine the two Aboxes together, and apply UCONN algorithm to the graph of the combined Abox.

If the graph is connected, then the two Aboxes depend on each other. We, then, add dep($\mathcal{A}_i, \mathcal{A}_j$) to the set depset as they are dependent, i.e., the function returns True. We get

$$\text{depset} = \{\text{dep}(\mathcal{A}_1, \mathcal{A}_2), \text{dep}(\mathcal{A}_1, \mathcal{A}_3), \text{dep}(\mathcal{A}_2, \mathcal{A}_3)\}$$

Since we know that $\mathcal{A}_1$ depends on $\mathcal{A}_2$, and also on $\mathcal{A}_3$, we virtually group them together, i.e., we will consider the data in both three Aboxes together. This is done by the combine_dependent_abox($\mathcal{A}$,depset) function. As a result we have $\mathcal{A}^g = \{\mathcal{A}_{123}, \mathcal{A}_4\}$, where $\mathcal{A}_{123} = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3$.

At this stage, we create source description for each Abox in $\mathcal{A}^g$, using the create_source_description($\mathcal{A}^g$) procedure:

- $SD(\mathcal{A}_{123}) = (\ DailyPlanet, Australia, Inquisitor, ENG, DKE, Chloe, MetroNews05, Clark, USA, BKKDelight, John, UQ)$
- $SD(\mathcal{A}_4) = (\ Malaysia, TsuHelp, WorldHelp, QuakeHelp)$

Suppose, we have a boolean query $John : (Parent \sqcap \exists hasChild.Employee)$. For every $\mathcal{A}_h \in \mathcal{A}^g$, we determine relevancy with respect to the query, using the procedure query_relevancy($SD(\mathcal{A}_h)$,*query*). The procedure will extract "John" from the query, and search whether "John" is in $SD(\mathcal{A}_h)$ or not. In this case, "John" is in $SD(\mathcal{A}_{123})$, but not in $SD(\mathcal{A}_4)$. Consequently, we simply query $\mathcal{A}_{123}$, using the instance_checking($\mathcal{A}_h$, *query*) function. The result from instance checking test of the query in $\mathcal{A}_{123}$ is True. Thus, answer = False $\vee$ True = True, which is the same result as when we query the whole Abox $\mathcal{A}$.

6 Complexity Analysis

Each Abox Graph can be trivially generated in $O(n^2)$, where n is the number of assertions in Abox $\mathcal{A}$. The next part is Abox dependency determination. We need $k(k-1)/2$ comparisons (2-combinations) of unordered pair of Aboxes, where k is the number of Aboxes in $\mathcal{A}$. Each comparison needs UCONN algorithm. UCONN can

be solved by DFS in $O(v^2)$, where v is the number of individuals in each Abox.

For source description, the create_source_description($\mathcal{A}^g$) procedure requires not more than $O(n^2)$ for all Aboxes, since it can be implemented using the quick sort algorithm. To determine the relevancy of an Abox to a query, we call the function query_relevancy($SD(\mathcal{A}_h)$,*query*) that operates in $O(n)$ for sequential search. Finally, we simply use the instance_checking($\mathcal{A}_h$,*query*) function to find the answer for each Abox, and simply combine the answer.

Till now, our space partitioning and reduction approach exploits at most PTIME algorithms in each part, i.e., the Abox dependency part and the source description part. Overall, our algorithm can be operated in PTIME, not including the instance checking part. Since the instance checking part is known to be solved in EXPTIME-complete, assuming P $\neq$ EXPTIME, the overall algorithm still operates in EXPTIME, but with a *reduced exponent*. The Abox dependency part will partition the search space, thus the exponent will be reduced if there are at least two partitions, e.g., the time complexity is reduced from 2^m to $2^n + 2^p$, where $m = n + p$. The source description part will further reduce the exponent if there are some Aboxes which can be eliminated from the process, e.g., the time complexity is reduced from 2^m to 2^q, where $q < m$.

7 The Extension

We intend to extend our work to non-boolean query answering (Abox retrieval) optimization. The space partitioning, Theorem 13, can be easily extended to cover Abox retrieval reasoning services. The main differences will be with the source description. Furthermore, we will investigate possible extensions of Theorem 13 to more expressive DLs.

In the nutshell, a source description of an Abox $\mathcal{A}$ for retrieval ($SD(\mathcal{A})$) consists of several types of source descriptions. In the basic setting, we shall have $SD^C(\mathcal{A})$ and $SD^R(\mathcal{A})$. $SD^C(\mathcal{A})$ is a set of *least common subsumer*s or *LCS*s of concepts appeared in concept membership assertions in the Abox $\mathcal{A}$. $SD^R(\mathcal{A})$ is a set of roles appeared in role membership assertions in the Abox $\mathcal{A}$. Let Q be a non-boolean query, C, D be concept expressions, and $\mathcal{A}_h$ be an Abox. The source description usage is different for each form of Q, for example:

- For the case $Q \leftarrow C$, $\mathcal{A}_h$ is relevant to Q if $\exists s \in SD^C(\mathcal{A}_h), Q \sqsubseteq s$
- For the case $Q \leftarrow C \sqcap D$, $\mathcal{A}_h$ is relevant to Q if $\mathcal{A}_h$ is relevant to C *and* $\mathcal{A}_h$ is relevant to D.
- For the case $Q \leftarrow C \sqcup D$, $\mathcal{A}_h$ is relevant to Q if $\mathcal{A}_h$ is relevant to C *or* $\mathcal{A}_h$ is relevant to D.
- For the case $Q \leftarrow \exists R.C$, $\mathcal{A}_h$ is relevant to Q if $R \in SD^R(\mathcal{A}_h)$ *and* $\mathcal{A}_h$ is relevant to C.
- For the case $Q \leftarrow \neg C$, $\mathcal{A}_h$ is relevant to Q if $\mathcal{A}_h$ is irrelevant to C.

These operations can be recursive. Hence, we can achieve a methodology for determine relevancy of the Abox $\mathcal{A}_h$ with respect to the arbitrary-formed non-boolean query Q. Note that the correctness of the above methodology follows from the independent Abox theorem, extended for non-boolean query.

8 Summary and Discussion

The optimization approach presented in this work is based on the procedure normally adopted for deduction over a description logic knowledge base (the query answering process over such knowledge base is a deduction process). In particular we refer to the tableaux algorithm. Traditionally, tableaux algorithm was designed to prove the satisfiability problem. The main idea behind this algorithm is based on a notational variant of the first order tableaux calculus. In fact, a tableaux algorithm tries to prove the satisfiability of a concept expression C by demonstrating a nonempty model of C. It constructively builds a model for a given concept. The process of constructing a model proceeds by completing a constraint system (Tessaris 2001), using a set of consistency-preserving completion (or expansion) rules. The process will continue if it can extend the existing constraint system. In $\mathcal{ALC}$ reasoning with $\mathcal{T}$ and $\mathcal{A}$, the process will proceed via a role membership assertion. The idea behind our work is to specify the condition where we guarantee that the reasoning process over $\mathcal{A}_1$ will never proceed to $\mathcal{A}_2$ if $\mathcal{A}_1$ and $\mathcal{A}_2$ are independent from each other. This optimization, in particular, the space partitioning part, can be seen as a divide-and-conquer technique. A general disadvantage of this kind of technique is the parts overlap. However, in this work we proposed a methodology to avoid overlapping part, thus, it does not suffer from such disadvantage of the divide-and-conquer technique.

Apparently, the major drawback of this approach is obviously the additional cost from Abox dependencies and source descriptions determination. But the cost is still in PTIME, as shown in previous section. One may argue that our space partitioning approach would support the reduction in apparent worst-case complexity for query answering, but at a high price in practice, in particular for a large knowledge base. However, this is not a scholarly argument, because the larger the knowledge base is, the less the relative cost is (recall that the normal query answering is in EXPTIME while the additional cost is in PTIME). Thus, our approach should behave well in practice. The only issue that we must give additional attention to is a design of effective Abox dependency information distribution, minimizing information exchange between nodes in the network, where each node represents an Abox. In addition, if the data pages do not change frequently, then there is no need to recompute the dependency of the Abox. In addition data source can be organised in indexes for fast retrieval.

One may also argue that this work is based on $\mathcal{ALC}$ description logic which is weaker than OWL. We accept this argument since at first we want to consider OWL-DL. However, OWL-DL is based on $\mathcal{SHOIQ}(\mathcal{D})$ description logic which is, in fact, the extension of $\mathcal{ALC}$. In the history of description logic, the least expressive language was investigated first. Then, the result were extended to cover more expressive languages consecutively. Consequently, we consider $\mathcal{ALC}$ first. We will, then, investigate the extension to cover OWL, in particular, OWL-DL, later.

This approach can be applied to a system which allows only one ontology (or Tbox). Though the Semantic Web technology tends to exploit multiple ontologies. In ontology-based integration of information area (Wache, Vögele, Visser, Stuckenschmidt, Schuster, Neumann & Hübner 2001), we can divide the exploitation of ontology into 3 approaches: single-ontology approach, e.g., SIMS (Arens, Hsu & Knoblock 1996), multiple-ontologies approach, e.g., OBSERVER (Mena, Kashyap, Sheth & Illarramendi 1996), and hybrid-ontology approach, e.g., COIN (Goh 1997). The single ontology approach allows only one ontology in the system while multiple-ontologies approach allows many ontologies in the system. The multiple-ontologies approach requires additional mapping specifications between each pair of ontologies. Since such mappings are infact ontologies themselves (Akahani, Hiramatsu & Kogure 2002), we need additional $n(n-1)/2$ ontologies for such an approach, where n is number of existing ontologies in the system. In hybrid-ontology approach, a global ontology and additional n mapping specifications (between global ontology and each local ontol-

ogy) are required. Hence the single-ontology approach can be viewed as generalization of the other two approaches. Thus, we follow such approach. In addition, since the aim of our work is to study how to query multiple data sources, thus we do not need to add complexity arisen from ontology mapping in the last two approaches. Simple single-ontology approach, but not trivial for query answering, is enough. Note that we can extend our work to include multiple ontologies later when the research about ontology binding and ontology mapping and ontology integration is more mature.

This approach can be applied to a system which allows multiple data sources (or Aboxes). We can think of an Abox as an RDF document. In addition RDF databases try to partition RDF triples in disjoint graphs , where each graph can be understood as a data page of our approach. However, recent research has shown that there are several semantic problems when people tried to layer an ontology language, i.e., OWL, on top of the RDF layer (Pan & Horrocks 2003). Such problems stem from some features/limitations of RDF, e.g., no restriction on the use of built-in vocabularies, and no restriction on how an RDF statement can be constructed since it is just a triple. Thus, this implies that the ontology layer may be not compatible with the RDF layer of the Semantic Web. However, there is a work proposing additional layer on top of the RDF layer, i.e., RDF(FA) (Pan & Horrocks 2003). This layer corresponds directly to Aboxes, thus RDF(FA) may be very useful in the future.

There are few works related to our work, i.e., Instance Store (Horrocks et al. 2004) and RACER (Haarslev & Möller 2002). Both works propose retrieval optimization techniques. Hence, our approach seems to be the first approach for Abox instance checking optimization. Instance Store imposes an unnatural restriction on Abox, i.e., enforcing Abox to be role-free. This is a severe restriction since role names are included even for $\mathcal{FL}^-$ ($\mathcal{ALC}$ without atomic negation), a DL with limited expressive power. RACER proposes several innovative Abox reasoning optimization techniques. However, RACER allows single Abox, while our approach allows multiple Aboxes. Thus after we apply our techniques to reduce the reasoning search space, we can apply RACER techniques to reduce it further. Consequently, the approach taken in RACER seems to be complementary to ours. We will investigate the combination of our approach and RACER approach in the future.

References

Akahani, J., Hiramatsu, K. & Kogure, K. (2002), Coordinating heterogeneous information services based on approximate ontology translation, *in* 'Proceedings of AAMAS-2002 Workshop on Agentcities: Challenges in Open Agent Systems', pp. 10–14.

Arens, Y., Hsu, C. & Knoblock, C. A. (1996), 'Query processing in the sims information mediator', *Advanced Planning Technology* .

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. & Patel-Schneider, P., eds (2003), *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, New York.

Baader, F. & Hollunder, B. (1991), A terminological knowledge representation system with complete inference algorithm, *in* 'The Workshop on Processing Declarative Knowledge', pp. 67–86.

Berners-Lee, T. (1999), *Weaving the Web : the Original Design and Ultimate Destiny of the World Wide Web by its Inventor*, HarperSanFrancisco, San Francisco.

Donini, F. M., Lenzerini, M., Nardi, D. & Nutt, W. (1991), The complexity of concept languages, *in* J. Allen, R. Fikes & E. Sandewall, eds, 'Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR-91)', Massachusetts, pp. 151–162.

Donini, F. M., Lenzerini, M., Nardi, D. & Schaerf, A. (1996), 'Reasoning in description logics', *Foundation of Knowledge Representation* pp. 191–236.

Donini, F. M. & Massacci, F. (2000), 'Exptime tableaux for $\mathcal{ALC}$', *Artificial Intelligence* **124**(1), 87–138.

Doyle, J. & Patil, R. (1991), 'Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services', *Artificial Intelligence Journal* **48**, 261–297.

Goh, C. H. (1997), Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Sources, PhD thesis, MIT.

Goodchild, A. (1998), Database Discovery in the Organizational Environment, PhD thesis, University of Queensland.

Haarslev, V. & Möller, R. (2002), Optimization strategies for instance retrieval, *in* 'Proceedings of the International Workshop on Description Logics (DL 2002)'.

Horrocks, I. (1997), Optimising Tableaux Decision Procedures for Description Logics, PhD thesis, University of Manchester.

Horrocks, I., Li, L., Turi, D. & Bechhofer, S. (2004), The instance store: Description logic reasoning with large numbers of individuals, *in* 'International Workshop on Description Logics (DL 2004)', pp. 31–40.

McGuinness, D. L., Fikes, R., Stein, L. A. & Hendler, J. (2003), Daml-ont: An ontology language for the semantic web, *in* D. Fensel, J. Hendler, H. Lieberman & W. Wahlster, eds, 'Spinning the Semantic Web: Bringing the World Wide Web to its Full Potential', MIT Press.

Mena, E., Kashyap, V., Sheth, A. & Illarramendi, A. (1996), Observer: An approach for query processing in global information systems based on interoperability between pre-existing ontologies, *in* 'Proceedings of the 1^{st} IFCIS: International Conference on Cooperative Information Systems (CoopIS '96)'.

Pan, J. Z. & Horrocks, I. (2003), Rdfs(fa): A dl-ised sub-language of rdfs, *in* 'Proceedings of the 2003 International Workshop on Description Logics (DL2003)'.

Schmidt-Schauß, M. & Smolka, G. (1991), 'Attributive concept descriptions with complements', *Artificial Intelligence* **48**(1), 1–26.

Stuckenschmidt, H. (2003), 'Query processing on the semantic web', *Künstliche Intelligenz* **17**.

Tessaris, S. (2001), Questions and Answers: Reasoning and Querying in Description Logic, PhD thesis, University of Manchester.

Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H. & Hübner, S. (2001), Ontology-based integration of information - a survey of existing approaches, *in* 'Proceedings of the IJCAI-01 Workshop: Ontologies and Information Sharing', pp. 108–117.