A Scalable Approach for Partitioning OWL Knowledge Bases

Yuanbo Guo and Jeff Heflin

Computer Science & Engineering Department, Lehigh University, Bethlehem, USA
{yug2, heflin}@cse.lehigh.edu

Abstract. We describe an approach to partitioning a large OWL ABox with respect to a TBox so that specific kinds of reasoning can be performed separately on each partition and the results trivially combined in order to achieve complete answers. The main features of our approach include: a reasonable tradeoff between the complexity of the task and the granularity of partitioning; worst-case polynomial time complexity; and the ability to handle problems that are too large for main memory. In addition, we show promising experimental results on both the Lehigh University Benchmark data and the real world FOAF data. This work could contribute to the development of scalable Semantic Web systems that need to deal with large amounts of data.

1 Introduction

OWL, the W3C recommended Web Ontology Language, is one of the key standards for building the Semantic Web. OWL has three species, i.e. OWL Lite, OWL DL and OWL Full, with increasing expressivity and thus increased reasoning complexity. It is well known that OWL Lite and OWL DL logically correspond to expressive Description Logics (DLs)[1] ($\mathcal{SHIF}$(D) and $\mathcal{SHOIN}$(D) respectively) [13]. Thus, from the logical point of view, an OWL Lite/DL knowledge base can be seen as consisting of a DL TBox $\mathcal{T}$ and a DL ABox $\mathcal{A}$. The TBox contains axioms about concepts (sets of objects) and roles (binary relations) while the ABox is a set of assertions about individuals (concept assertions) and assertions about their relationships (role assertions).

Reasoning with OWL is highly complex. Even OWL Lite has exponential worst-case complexity. Nonetheless, contemporary reasoners such as FaCT [11], Racer [8] and Pellet [14] tend to work well with realistic TBoxes, which usually have moderate sizes. All of the systems are main memory-based systems and are based on highly optimized implementation of the tableaux algorithms.

Recently there is a growing interest for practical ABox reasoning (e.g. [9] and [12]). As Horrocks et al. [12] pointed out, a great challenge for systems to support ABox reasoning is due to the fact that ABoxes can be extremely large in a setting like the Semantic Web. In fact, statistics from the well known Semantic Web search engine SWOOGLE [2] have already revealed the trend that, as the Semantic Web progresses, instance data will greatly outnumber ontologies. According to SWOOGLE, which has thus far indexed over one million Semantic Web documents,

[1] This paper assumes readers' basic knowledge about DLs.

the ratio of data documents to ontologies was about 80 to 1 in 2005, and this gap has widened by 40% in 2006.

This poses a great challenge for the development of practical Semantic Web systems. Even if we disregard reasoning complexity, resource limitations (e.g. physical memory) could prevent the system from handling a large ABox. To address the problem, it is worthwhile to investigate the issue along different lines. For instance, research on reasoning techniques and their optimized implementation will certainly be helpful. Also there have been some preliminary proposals [14] for implementing secondary-storage support for reasoning with large ABoxes, although the performance impact is yet to be known.

In this work, we consider addressing the issue from a different perspective. Our primary goal is to develop an approach to help existing systems overcome memory limitations so that they will be able to handle a large ABox that is originally beyond their capabilities. To this end, we adopt a divide-and-conquer approach, that is, we consider how to partition an OWL ABox into smaller pieces that could be processed separately while still guaranteeing the completeness of the reasoning in question when the results are combined.

In what follows, first we formally define the kind of partitioning we are concerned with. Then we describe our partitioning approach. Following that, we present an empirical evaluation.

2 Independent ABox Partitioning

Amir and McIlraith [1] have defined partitioning of a logical theory. We have specialized the definition for an ABox, as follows:

Definition 1 (ABox Partitioning). $\{\mathcal{A}_i\}_{i\leq n}$ *is a partitioning of ABox* $\mathcal{A}$ *iff* $\mathcal{A} = \cup_i \mathcal{A}_i$.

Of course not every partitioning is useful to us. Recall that our goal is that we can work on each of the partitions independently while still guaranteeing complete reasoning if the results are combined. Specifically, in this work, we focus on instance checking, a fundamental reasoning task with respect to an ABox. In other words, we are concerned with the inference of concept assertions (in the form of $a{:}C$) and the inference of role assertions (in the form of $\langle a,b\rangle{:}R$). Thus a partitioning that meets our requirements, which we shall call an independent partitioning, is as follows.

Definition 2 (Independent ABox Partitioning). *Given an OWL knowledge base* $\mathcal{K}=(\mathcal{T},\mathcal{A})$*, a partitioning of* $\mathcal{A}$*,* $\{\mathcal{A}_i\}_{i\leq n}$*, is an independent partitioning of* $\mathcal{A}$ *with respect to* $\mathcal{T}$ *iff for every concept assertion or role assertion* φ *such that* $\mathcal{K}\models\varphi$*, there exists* $\mathcal{A}_i$ *such that* $(\mathcal{T},\mathcal{A}_i)\models\varphi$.

Note that in previous work [6], we have discussed the notion of independency between OWL ABoxes and the detection of such relationship in some special cases. This work can be seen as complementary, i.e., we consider how to develop a general partitioning approach that can generate independent ABoxes.

Given Definition 2, there could potentially be a range of suitable partitions with varying granularities. We identify some of them here. 1) *The most fine-grained partitioning* is an independent partitioning that in addition guarantees the smallest possible size of each partition. 2) *Minimal set based partitioning*. This is an

independent partitioning $\{\mathcal{A}_i\}_{i\leq n}$ that in addition satisfies: for every $\mathcal{A}_i$ there exists a concept assertion or role assertion φ such that $\mathcal{A}_i$ is a minimal set with the property $\mathcal{A}_i \models \varphi$. This partitioning does not take into account that multiple minimal sets may entail the same assertion. 3) *Common individual names based partitioning*. This is a partitioning such that all the partitions are disjoint in individual names and thus is independent [6].

Among the above kinds of partitioning, 3) is simple to implement but the granularity may not be satisfactory. On the other hand, as can be imagined, to achieve a partitioning as fine-grained as 1) or 2) essentially amounts to a task with no less computational complexity than complete reasoning on the ABox. This is just what we wanted to avoid. In light of that, in this work, we have aimed at an approach that generates partitioning that falls in between 2) and 3) and at the same time has a polynomial worst-case time complexity.

3 An Approach for Independent ABox Partitioning

Now we discuss our approach to determining independent ABox partitions. As the first step of the work, we will focus on OWL Lite. From the logical point of view, we will consider OWL files that correspond to a DL $\mathcal{SHIF}$ knowledge base. In addition, we will focus on the case where the assertion φ in Definition 2 is either a role assertion or a concept assertion whose concept is atomic. Moreover, in order to concentrate on the main idea, we ignore datatypes, which can easily be supported by some adaptations. For the same reason, we assume there are no (in)equality assertions in the ABox. Again, our approach can be easily adapted to address the presence of such assertions.

3.1 Determining Relevant Assertions

In order to determine partitions, a core question facing us is to decide relevant assertions in the ABox in the sense that they together might derive a new inference. To this end, we have utilized the natural deduction-style inference rules.

Royer and Quantz [15, 16] have developed a generic approach to deriving a sound and complete set of inference rules for any given DL. Their approach utilizes a sound and complete Sequent Calculus for FOL (See [4] for the proof of soundness and completeness). In order to derive rules for a given DL, they translate that DL into FOL according to the well-known method and then identify from the resultant FOL formulae necessary and sufficient conditions of provability in the Sequent Calculus. These conditions will in turn lead to the derivation of inference rules for the DL. In their work [16], they have applied this approach to a very expressive DL, which includes constructs such as intersection, negation, value restriction, existential quantity, role hierarchy, transitive role, inverse role and number restriction, and thus subsumes $\mathcal{SHIF}$. The resultant set of rules they have given is complete if nominals are ignored, but this does not affect us since $\mathcal{SHIF}$ does not have such a feature.

Based on their results, we have identified the following set of inference rules over DL $\mathcal{SHIF}$ knowledge base $(\mathcal{T},\mathcal{A})$. (In the rules, *a*, *b*, *c* are any individual names, *C* any concept expression, and *R* any role name.)

TBox Rules

Subsumptions in the TBox will affect reasoning on ABox individuals (see R3 and R7). There are over 100 TBox rules for inferring concept subsumptions and role subsumptions. To save space we do not list them here and refer readers to [16]. (Note due to this, some DL constructs such as negation and disjunction do not explicitly appear here.) Below we use $\mathcal{T}\vdash C_1 \sqsubseteq C_2$ or $\mathcal{T}\vdash R_1 \sqsubseteq R_2$ to denote such inferences.

ABox Rules

For ABoxes rules, we use $\mathcal{A}\vdash\varphi$ as an abbreviation for $(\mathcal{T},\mathcal{A})\vdash\varphi$. Also for simplicity we ignore the constructs of $\geq 0R$, $\geq 1R$ and $\leq 0R$ since they can be translated into $\top$, $\exists R.\top$ and $\forall R.\bot$ respectively.

R1) If $\varphi\in\mathcal{A}$ then $\mathcal{A}\vdash\varphi$
R2) $\mathcal{A}\vdash a:\top$
R3) If $\mathcal{T}\vdash C_1\sqsubseteq C_2$ and $\mathcal{A}\vdash a:C_1$ then $\mathcal{A}\vdash a:C_2$
R4) If $\mathcal{A}\vdash a:C_1$, $a:C_2$ then $\mathcal{A}\vdash a:C_1\sqcap C_2$
R5) If $\mathcal{A}\vdash a:\forall R.C$, $\langle a,b\rangle:R$ then $\mathcal{A}\vdash b:C$
R6) If $\mathcal{A}\vdash\langle a,b\rangle:R$, $b:C$ then $\mathcal{A}\vdash a:\exists R.C$
R7) If $\mathcal{T}\vdash R_1\sqsubseteq R_2$ and $\mathcal{A}\vdash\langle a,b\rangle:R_1$ then $\mathcal{A}\vdash\langle a,b\rangle:R_2$
R8) If $\mathcal{A}\vdash\langle a,b\rangle:R$ then $\mathcal{A}\vdash\langle b,a\rangle:R^-$
R9) If $R\in\boldsymbol{R}_+$(transitive role) and $\mathcal{A}\vdash\langle a,b\rangle:R$, $\langle b,c\rangle:R$ then $\mathcal{A}\vdash\langle a,c\rangle:R$
R10) If $\mathcal{A}\vdash a:\leq 1R$, $\langle a,b_1\rangle:R$, $\langle a,b_2\rangle:R$ then $\mathcal{A}\vdash b_1=b_2$
R11) If $\mathcal{A}\vdash a=b$ and $a:C$ (resp. $\langle a,c\rangle:R$, $\langle c,a\rangle:R$, $a=c$) then $\mathcal{A}\vdash b:C$ (resp. $\langle b,c\rangle:R$, $\langle c,b\rangle:R$, $b=c$)

The above rules may not be suitable for implementing an effective reasoning system given their forward-chaining style. However, the advantage is that they provide us with an intuitive guidance for partitioning: *generally, assertions in the antecedent of an inference rule should be placed in the same partition*. However, what complicates things is that those assertions could be implicitly inferred statements and consequently we have to pinpoint their antecedent assertions in the ABox. This essentially requires something akin to backward-chaining reasoning on the whole knowledge base, which is not feasible for our purpose.

Given this, we have decided to make some changes to the rules so that they are much easier for us to determine which assertions might infer which others and decide partitions accordingly. *In terms of reasoning, the adapted rules are still complete but will be unsound. However, as far as partitioning is concerned, the unsoundness will not affect the correctness of a partitioning and the only consequence of the unsoundness will be that we potentially sacrifice the granularity of partitioning, i.e., we allow the case where two assertions are required to be in the same partition although they are indeed irrelevant.* The new rules are listed below. To distinguish from the original one, we denote this new inference by |~ and we call it "might infer".

TBox Rules

TR1') If $\mathcal{T}\vdash R_1\sqsubseteq R_2$ then $\mathcal{T}\mid\sim R_1\sqsubseteq R_2$
TR2') For any concepts C_1 and C_2, $\mathcal{T}\mid\sim C_1\sqsubseteq C_2$

ABox Rules

R1') If $\varphi\in\mathcal{A}$ then $\mathcal{A}\mid\sim\varphi$
R2') $\mathcal{A}\mid\sim a:\top$
R3') If $\mathcal{T}\mid\sim C_1\sqsubseteq C_2$ and $\mathcal{A}\mid\sim a:C_1$ then $\mathcal{A}\mid\sim a:C_2$
R4') If $\mathcal{A}\mid\sim a:C_1$, $a:C_2$ then $\mathcal{A}\mid\sim a:C_1\sqcap C_2$

R5') If $\forall$-Possible($\mathcal{T}$, R) and $\mathcal{A}|\sim a{:}\forall R.C$, $\langle a,b\rangle{:}R$ then $\mathcal{A}|\sim b{:}C$

R6') If $\exists$-Useful($\mathcal{T}$, R) and $\mathcal{A}|\sim\langle a,b\rangle{:}R$, $b{:}C$ then $\mathcal{A}|\sim a{:}\exists R.C$

R7') If $\mathcal{T}|\sim R_1 \sqsubseteq R_2$ and $\mathcal{A}|\sim\langle a,b\rangle{:}R_1$ then $\mathcal{A}|\sim\langle a,b\rangle{:}R_2$

R8') If $\mathcal{A}|\sim\langle a,b\rangle{:}R$ then $\mathcal{A}|\sim\langle b,a\rangle{:}R^-$

R9') If $R\in \boldsymbol{R}_+$, $\mathcal{A}|\sim\langle a,b\rangle{:}R, \langle c,d\rangle{:}R$, and $\mathcal{A}|\sim a{=}c$ or $\mathcal{A}|\sim a{=}d$ or $\mathcal{A}|\sim b{=}c$ or $\mathcal{A}|\sim b{=}d$ then $\mathcal{A}|\sim\langle a,c\rangle{:}R$, $\langle a,d\rangle{:}R$, $\langle b,c\rangle{:}R$, $\langle b,d\rangle{:}R$

R10') If $\leq$1-Possible($\mathcal{T}$, R) and $\mathcal{A}|\sim a{:}\leq 1R$, $\langle a,b_1\rangle{:}R$, $\langle a,b_2\rangle{:}R$ then $\mathcal{A}|\sim b_1{=}b_2$

R11') If $\mathcal{A}|\sim a{=}b$ and $\mathcal{A}|\sim a{:}C$ (resp. $\mathcal{A}|\sim\langle a,c\rangle{:}R$ or $\mathcal{A}|\sim\langle c,a\rangle{:}R$ or $\mathcal{A}|\sim a{=}c$) then $\mathcal{A}|\sim b{:}C$ (resp. $\mathcal{A}|\sim\langle b,c\rangle{:}R$ or $\mathcal{A}|\sim\langle c,b\rangle{:}R$ or $\mathcal{A}|\sim b{=}c$)

We now look at the major changes and their implications.

TR1') For role subsumptions, we have decided not to define concrete rules. Instead, as we will show in the next subsection, our partitioning algorithm will make use of a complete reasoner to compute the subsumption order ($\sqsubseteq$) of the roles in $\mathcal{T}$. Note we can do this easily since all roles are atomic in $\mathcal{SHIF}$.

TR2') Concept subsumptions are more complicated since a concept may be a complex one (e.g. $\forall R_1.C_1 \sqcap \exists R_2.C_2$). As such, just knowing the subsumption order of atomic concepts in the TBox is not sufficient. To reduce the complexity, we have decided to assume that the TBox "might infers" any concept subsumptions.

R3') Although this rule is virtually identical to R3, when combined with TR2', R2' and R4', it has significant implications. In particular, for every concept expression C and individual a, $\mathcal{A}|\sim a{:}C$. As will be seen later, this will impact partitioning in the following way: for any individual, all of its concept assertions in the ABox as well as other assertions that are necessary for the inference of that individual's concept memberships will be assigned to the same partition.

Given this rule allows $\mathcal{A}|\sim a{:}C$ for any a and C, it is reasonable to ask why we still need other rules that may infer the same assertions (e.g. R5' and R6'). The answer is that the purpose in defining the $|\sim$ rule set here is not for doing reasoning per se. Instead, it will be used as guidance to the partitioning algorithm we develop later. $|\sim$ is complete but not sound (see Proposition 1 below). Therefore, even when two different sets of assertions constitute the proof for the same inference through different $|\sim$ rules, we still need to guarantee that for each set, the assertions it contains are put into the same partitions.

R5') With this rule, we want to eliminate the complexity of having to exactly determine whether a is an instance of $\forall R.C$. We instead try to determine, by only taking into account certain information from $\mathcal{T}$, whether it is possible that an individual is inferred as of some concept $\forall R.C$ ($C{\neq}\top$). We use $\forall$-Possible to denote such a property of R. We define:

$\forall$-Possible($\mathcal{T}$, R) is true if a concept $\forall S.C$ occurs on the right hand side of some general concept inclusion (GCI) in $\mathcal{T}$ and $R \sqsubseteq S$. (via Royer's Rule 12)

(Note here $\sqsubseteq$ denotes the subsumption relationship, which subsumes the reflexive-transitive closure of the role hierarchy in $\mathcal{T}$. Also note trivially $C_1 \equiv C_2$ can be translated into two GCIs.)

We have derived the above definition through two ways. First, we resort to a syntactical analysis of the TBox. For instance, if there is a GCI $C_1 \sqsubseteq \forall R.C_2$ in $\mathcal{T}$, then $\forall$-Possible($\mathcal{T}$, R) is true. This is straightforward because, given an ABox supporting $a{:}C_1$, we will be able to infer that $a{:}\forall R.C_2$. Second, we analyze the above mentioned TBox inference rules provided by Royer and Quantz [16]. As a result, the *"$R \sqsubseteq S$"* condition in the definition stemmed from the following subsumption rule: $C_1 \sqsubseteq C_2$,

$R_2 \sqsubseteq R_1 \rightarrow \forall R_1.C_1 \sqsubseteq \forall R_2.C_2$ (Rule 12). The rationale is as follows: suppose $C_1 \sqsubseteq C_2$ and $R_2 \sqsubseteq R_1$. If $\forall$-Possible($\mathcal{T}$, R_1) is true, then according to the notion of $\forall$-Possible, it is possible that some individual a is of type $\forall R_1.C_1$. Thus by the rule we get $a: \forall R_2.C_2$, which in turn means $\forall$-Possible($\mathcal{T}$, R_2) is also true. Similar analysis is used below, but to save space, we will just reference the rules we have used.

R6') Originally, the implication of R6) is that $\langle a,b\rangle:R$ and $b:C$ are always relevant assertions: considering Definition 2, suppose φ be $a: \exists R.C$. In order to guarantee the partitions to be complete with respect to φ, we would have to put $\langle a,b\rangle:R$ and $b:C$ in the same partition. However, as mentioned earlier, in this work we focus on the case that if φ is a concept assertion $a:C$, C is atomic. Therefore, in adapting the rule, we have utilized that fact in order to help refine the granularity of partitioning. The idea is that if $a: \exists R.C$ may help infer that a belongs to another concept (including an atomic one), then $\langle a,b\rangle:R$ and $b:C$ should be considered relevant; otherwise, the rule needs not to be applied. As with R5'), the precise characterization is too complex and thus instead we try to determine, based on some information of $\mathcal{T}$ only, whether some fact $a: \exists R.C$ could possibly help infer that a belongs to another concept. For example, if in $\mathcal{T}$ we have $\exists R.C_1 \sqsubseteq C_2$ or $\exists R.C_1 \sqsubseteq \forall R.C_2$ then the answer is yes. We denote the above property by $\exists$-Useful and have derived its rule using a similar method to R5').

$\exists$-Useful($\mathcal{T}$, R) is true if any of the following is true:

1) A concept $\exists S.C$ occurs on the left hand side of some GCI in $\mathcal{T}$ and $R \sqsubseteq S$. (via Royer's Rule 13)
2) There exists S such that $\forall$-Possible($\mathcal{T}$, S) is true and $R \sqsubseteq S$. (via Royer's Rule 18)

R9') For reasoning concerning transitivity, the complexity lies in that we have to match the object of one assertion with the subject of another in order to derive a new assertion, and in addition to worry about which individuals appear in which positions in the inferred assertion. We have given up this restriction in the new rule. One benefit we get from that is now we could immediately tell what role assertions might be inferred from a set of assertions of R. In addition, based on R7'–R9', we are able to conduct a simple individual names based partitioning for assertions related to R. This will be shown in the next subsection.

R10') The idea is similar to R5'). To eliminate the reasoning about $a: \leq 1R$, we have introduced $\leq$1-Possible($\mathcal{T}$, R), which is a judgment, based on certain information from $\mathcal{T}$, that it is possible that an individual is inferred as of concept $\leq 1R$.

$\leq$1-Possible($\mathcal{T}$, R) is true if a concept $\leq 1S$ occurs on the right hand side of some GCI in $\mathcal{T}$ and $R \sqsubseteq S$. (via Royer's Rule 40)

From the above, we can see that $\forall$-Possible, $\exists$-Useful and $\leq$1-Possible play a role in restricting the potential loss of the granularity of partitioning due to the relaxation of other rules.

Proposition 1. $|\sim$ *is complete but not sound with respect to the inference of role assertions and concept assertions whose concept is atomic.*

Proof. Let φ be an assertion of the above kinds. We know $\mathcal{K} \models \varphi \Leftrightarrow \mathcal{K} \vdash \varphi$. From the description of $|\sim$, we can see that if $\mathcal{K} \vdash \varphi$ then $\mathcal{K} |\sim \varphi$. Specifically, among the rules, TR1', R1'–R4', R7', R8' and R11' are virtually identical to the original ones. For TR2' and R9'), their preconditions and/or consequences are relaxed. Thus completeness is preserved. As to R5', R6' and R10', they actually have strengthened

preconditions. However, this will not affect completeness. For R5', as long as $a{:}\forall R.C$ is entailed, $\forall$-Possible($\mathcal{T}$, R) will always be true, and thus the precondition is still satisfiable. Similarly for R10'. For R6', the notion of $\exists$-Useful determines that if $\exists$-Useful($\mathcal{T}$, R) is false, excluding R6') will not affect the final inference of any $a{:}C$ where C is atomic, as well as any possible inferences about other individuals through other rules. The above shows $|\sim$ is complete. On the other hand, it is easy to find an example such that $\mathcal{K}|\sim\varphi$ but not $\mathcal{K}\models\varphi$, thus showing that the rules are not sound.

Overall, with $|\sim$, we have transformed the inference rules into a set that can better serve our purpose so that we could develop a partitioning algorithm upon it.

3.2 Partitioning Rules and Algorithms

3.2.1 Constructing Chunk Graph

Our partitioning algorithm consists of two major steps:
1) Given the knowledge base, construct a chunk graph for its ABox.
2) Determine the final partitions based on the resultant chunk graph.

First, we introduce chunks and chunk graph as the building block of partitions.

Definition 3 (Chunk and Chunk graph). *A chunk graph $G=(V, E)$ for an ABox $\mathcal{A}$ is as follows: 1) Each vertex of G represents a chunk, which is a subset of $\mathcal{A}$; chunks on G are disjoint and their union is $\mathcal{A}$. 2) G is a directed graph and $(ck_1,ck_2)\in E$ means that a partition P containing chunk ck_2 must also contain chunk ck_1.*

To construct the chunk graph, we define a set of basic rules (called chunk rules) and a procedure for applying those rules. Most of the rules are derived by referring to one or more of the might infer ($|\sim$) rules and the procedure designates the application of chunk rules by taking into account the interaction between the inference rules.

ChunkRule 1: (Based on TR2', R2'–R4') Create a chunk for each individual a in $\mathcal{A}$: $\{a{:}C\}_{a:C\in\mathcal{A}}$. Hereafter we use *chunk*(a) to denote the chunk that holds the concept assertions of a. Recall an assertion can only be in one chunk. Also note *chunk*(a) is initially empty if a is untyped (although this is not allowed by OWL Lite.)

ChunkRule 2: Create a chunk for each role assertion φ in $\mathcal{A}$. Hereafter we will use *chunk*(φ) to denote the chunk that contains φ. Trivially, *chunk*(φ) $|\sim \varphi$.

ChunkRule 3: (Based on R10', R11') If $\leq$1-Possible($\mathcal{T}$, R) and $ck_1|\sim\langle a,b_1\rangle{:}R$, $ck_2|\sim\langle a,b_2\rangle{:}R$ then
1) Merge ck_1 and ck_2 to ck. (Note ck_1 and ck_2 can be the same chunk.)
2) Merge *chunk*(b_1) and *chunk*(b_2) to ck'.
3) Draw an arc from *chunk*(a) to ck.
4) Draw an arc from ck to ck'.
5) For every role assertion φ involving b_1 or b_2, draw an arc from ck to *chunk*(φ).
6) Record the information $b_1\approx b_2$ (abbreviation for $\mathcal{A}|\sim b_1=b_2$).

ChunkRule 4: (Based on R7'–R9') For each $R\in\boldsymbol{R}_+$, conduct an individual names based merging on the following set of assertions

$\{\langle a,b\rangle{:}R_1 \mid R_1\sqsubseteq R$ or R_1 is the inverse of $R\}$ (Note $R\sqsubseteq R$)

according to the following principle:

for any two of the above assertions $\varphi_1=\langle a,b\rangle{:}R_1$ and $\varphi_2=\langle c,d\rangle{:}R_2$, where $a\approx c$ or $a\approx d$ or $b\approx c$ or $b\approx d$, merge *chunk*(φ_1) and *chunk*(φ_2).

Recall we have relaxed the precondition of R9'. By virtue of that, we are able to do the above simple operation. Moreover, the relaxation of the consequence of the same rule has enabled us to quickly decide the assertions that the merged chunk might infer, as follows:

For a chunk *ck* having $R \in \boldsymbol{R}_+$, let

subject_list = {*a* | There exists $\langle a,b\rangle:R_1 \in ck$
where $R_1 \sqsubseteq R$ *or* $\langle b,a\rangle:R_1 \in ck$ where R_1 is the inverse of R}

object_list = {*a* | There exists $\langle b,a\rangle:R_1 \in ck$
where $R_1 \sqsubseteq R$ *or* $\langle a,b\rangle:R_1 \in ck$ where R_1 is the inverse of R}.

Then we know $ck \mid\sim \{\langle a,b\rangle:R \mid a \in subject\_list$ and $b \in object\_list\}$

ChunkRule 5: (Based on R5') If $\forall$-Possible($\mathcal{T}$, R) and $ck \mid\sim \langle a,b\rangle:R$ then
1) Draw an arc from *chunk*(*a*) to *chunk*(*b*).
2) Draw an arc from *ck* to *chunk*(*b*).

ChunkRule 6: (Based on R6') If $\exists$-Useful($\mathcal{T}$, R) and $ck \mid\sim \langle a,b\rangle:R$ then
1) Draw an arc from *chunk*(*b*) to *chunk*(*a*).
2) Draw an arc from *ck* to *chunk*(*a*).

---

procedure BUILD_CHUNK_GRAPH($\mathcal{K}$)
Input: OWL knowledge base $\mathcal{K}=(\mathcal{T},\mathcal{A})$
Output: a chunk graph *G* for $\mathcal{A}$, empty initially.
Local variables: *eq* keeps all the $\approx$ information, nil initially.

Process $\mathcal{T}$: 1) Use a complete reasoner to compute role subsumptions and role *inverseOf* relationships. 2) Do a syntactical analysis of $\mathcal{T}$ to determine the truth values of $\forall$-Possible, $\exists$-Useful and $\leq$1-Possible for each role.

Update *G* by applying ChunkRules 1 and 2.
repeat
Update *G* by applying ChunkRule 4.
Update *G* and *eq* by applying ChunkRule 3.
until neither *G* nor *eq* has been changed.
Update *G* by applying ChunkRules 5 and 6.
return *G*
end

Example 1. Let $\mathcal{K}=\langle\mathcal{T},\mathcal{A}\rangle$ wherein

$\mathcal{T}$ = {*Student* $\sqsubseteq \leq$1*advisor*,
Chair $\equiv \exists$ *headOf.Department*,
Department $\sqsubseteq \forall$ *subOrg.University*},

$\mathcal{A}$ = {*u1*:*University*, *d1*:*Department*, *g1*:*Group*, *g2*:*Group*,
f1:*Faculty*, *f2*:*Person*, *f3:Faculty, f4:Faculty, s1:Student,*
$\langle$*f3,d1*$\rangle$*:headOf,*
$\langle$*s1,f1*$\rangle$*:advisor,* $\langle$*s1,f2*$\rangle$*:advisor,*
$\langle$*f3,f1*$\rangle$*:worksFor,* $\langle$*f2,f4*$\rangle$*:worksFor,*
$\langle$*g1,d1*$\rangle$*:subOrg,* $\langle$*g2,d1*$\rangle$*:subOrg,* $\langle$*d1,u1*$\rangle$*:subOrg*}, and

subOrg $\in \mathbf{R}_+$, *worksFor* $\in \mathbf{R}_+$.

From $\mathcal{T}$ we can determine that $\leq$1-Possible($\mathcal{T}$, *advisor*), $\forall$-Possible($\mathcal{T}$, *subOrg*), $\exists$-Useful($\mathcal{T}$, *subOrg*), and $\exists$-Useful($\mathcal{T}$, *headOf*) are true. Figs. 1 and 2 illustrate how the chunk graph is constructed.

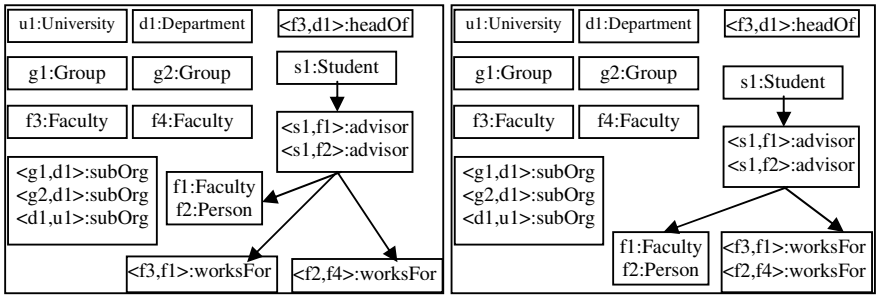

Fig. 1. a) Chunks are initialized by applying ChunkRules 1 and 2. Then ChunkRules 4 (for *subOrg*) and 3 (for *advisor*) were applied, and *f1≈f2* is recorded.

Note now *ck* |~{<*g*1,*d*1>:*subOrg,* <*g*1,*u*1>:*subOrg,* <*g*2,*d*1>:*subOrg,* <*g*2,*u*1>:*subOrg,* <*d*1,*d*1>:*subOrg,* <*d*1,*u*1>:*subOrg*},

ck being the chunk that contains those *subOrg* assertions.

b) ChunkRule 4 was applied for *worksFor* due to the new information of *f1≈f2*.

Now *ck* |~{<*f*3,*f*1>:*worksFor*, <*f*3,*f*4>:*worksFor*, <*f*2,*f*1>:*worksFor*, <*f*2,*f*4>:*worksFor*}, *ck* being the chunk that contains those *worksFor* assertions.

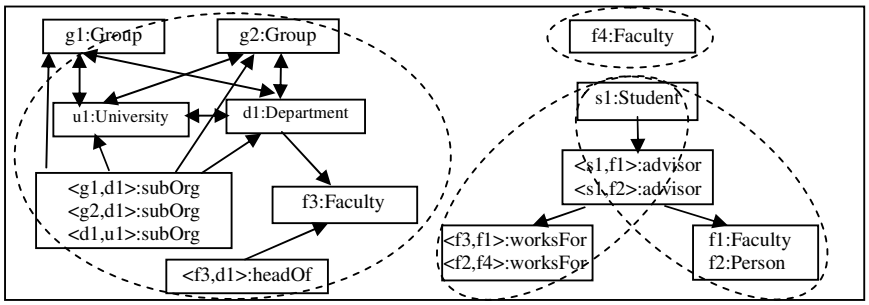

Fig. 2. ChunkRules 5 (for *subOrg*) and 6 (for *subOrg* and *headOf*) were applied and consequently more arcs were added. The circles are explained in Section 3.2.2.

This algorithm is guaranteed to terminate. First, there is a maximum number of elements in *eq* and *eq*'s size increases monotonically. Also there is a maximum number of arcs on *G*. Moreover, if we look at merging two chunks as adding two arcs between them, then the number of chunks is fixed after applying ChunkRules 1 and 2. Following that the arcs on *G* will increase monotonically. Thus there is a maximum number of changes to *eq* and *G*. The worst-case time complexity of the algorithm is polynomial in *n*, the size of $\mathcal{A}$. Specifically, since we do not need to compute concept subsumptions, processing the TBox can be done in a polynomial time. The complexity of applying each chunk rule is polynomial in *n* too. In addition, the number of chunks on *G* is no more than *n* and the number of edges no more than n^2. Moreover, the maximum size of *eq* is n^2. Overall, the complexity is polynomial in *n*.

Recall that the might infer (|~) rules are intentionally made unsound. *In terms of building a chunk graph, the consequence of the unsoundness will be that we might put two irrelevant assertions into the same chunk or we might add a path between two*

irrelevant chunks. As a result, we might get a coarser granularity of partitioning. Nevertheless, as can be seen, compared to the original rules, the might infer rules have greatly reduced the complexity and facilitated the decision making.

Proposition 2. *Let $\mathcal{K}=(\mathcal{T},\mathcal{A})$ be an OWL Lite knowledge base and G= BUILD_CHUNK_GRAPH($\mathcal{K}$). $\{\mathcal{A}_i\}_{i\leq n}$ is an independent partitioning of $\mathcal{A}$ (with respect to the inference of role assertions and concept assertions whose concept is atomic) if it complies with the semantics of G as defined by Definition 3.*

Proof (Sketch). Let φ be a role assertion or a concept assertion whose concept is atomic and $\mathcal{K}\models\varphi$. From the above description we can see that the way whereby the chunk rules and the algorithm are designed guarantees the following property. Suppose S is a minimal subset of $\mathcal{A}$ such that $(\mathcal{T},S)\models\varphi$, then there exists a chunk ck that satisfies: for every assertion $\beta\in S$, either β is in ck or there is a directed path from $Chunk(\beta)$ to ck. This means $C\supseteq S$, C being the union of all the chunks that can reach ck (including ck itself). Thus $(\mathcal{T},C)\models\varphi$. If the partitioning complies with the semantics of G, there must be an $\mathcal{A}_i$ such that $\mathcal{A}_i\supseteq C$. Therefore $(\mathcal{T},\mathcal{A}_i)\models\varphi$. Proved.

3.2.2 Determining Partitions

After chunk graph G is built, to determine the partitioning off G becomes fairly straightforward. We name this procedure CONSTRUCT_PARTITIONS(G).

---

procedure CONSTRUCT_PARTITIONS(G)
  Input: The chunk graph created for OWL knowledge base $\mathcal{K}=(\mathcal{T},\mathcal{A})$
  Output: an independent partitioning P of $\mathcal{A}$, nil initially.

  Compact G by merging every set of chunks that form a strongly connected component of G.
  For every chunk ck that has no outgoing links:
    Create a partition $p = \cup_i ck_i$ where ck is reachable from ck_i on G (including ck).
    $P = P \cup \{p\}$
  return P
end

In the above procedure, the most complex operations are those typical operations on the graph, for example computation of reachability between vertices, which has $O(n^3)$ worst-case complexity. Therefore, suppose n is the size of $\mathcal{A}$, the algorithm will have a time complexity of $O(n^3)$ in the worst situation where the number of chunks in the resultant graph equals to n and moreover the graph is dense.

Example 1 (continued). By the above algorithm, we will get 4 partitions, each enclosed by a dashed circle in Fig. 2. Note that chunks in multiple circles are contained in multiple partitions.

The above algorithm keeps the resultant partitions as small as possible by allowing chunks in the same partition only when they are required by the chunk graph. In practice, however, we may prefer a restricted number of partitions as long as their sizes do not go beyond a predefined limit. In that case, we can do a merging of the resultant partitions.

4 Implementation and Empirical Evaluation

We want to emphasize that our algorithm allows for an implementation based on secondary storage so as to accommodate large data. Specifically, we use MySQL as the backend DBMS. The input data are stored in the database in the form of triples. Moreover, intermediate results during the processing, such as the chunks, chunk graph, partitions, as well as relevant $\approx$ and $|\sim$ information, are all maintained in the database. Furthermore, one feature of the algorithm is that, during the process, it often only needs to work on a small subset of the input data at a time, for instance, concept assertions for the same individual, role assertions for the same role, an individual chunk, and so on. This locality property of the algorithm is important for the implemented system to be able to scale.

We have done experiments on both the synthetically created Lehigh University Benchmark data and the realistic FOAF data. First, we have made use of the Lehigh University Benchmark (LUBM) [7], which is a benchmark for evaluating OWL knowledge base systems with a focus on ABox processing. Using the data generator for LUBM, we have created test sets of a range of sizes from 1M to 13M.[2] The benchmark data commits to a realistic ontology of the university domain, which features *someValuesFrom*, *TransitiveProperty*, *inverseOf* and so on. Table 1 and Fig. 3 show the test results. As can be seen, the system scales well.

Table 1. Test results on LUBM data

Test Set (# of triples)	Partition Time (hh:mm:ss)	# of Partitions	Average Partition Size (# of triples)	Min Partition Size (# of triples)	Max Partition Size (# of triples)
1,311 K	00:21:18	396,197	21.2	1	1141
2,771 K	00:59:13	836,608	21.7	1	1154
6,863 K	03:22:05	2,071,365	21.6	1	1154
13,824 K	08:23:21	4,170,788	21.7	1	1184

(Test Environment: Windows XP OS; P4 2.8GHz CPU; 1G B of RAM; JDK 1.5)

Note that according to an experiment by Haarslev, Möller and Wessel [10], Racer, one of the representative OWL reasoners, would at that time not be able to handle the above sizes of LUBM datasets. (Their test was done on a P4 2.8GHz CPU 1GB RAM machine running Linux.) However, given a partition of a size of 5000 triples or less, Racer would be able to finish loading and ABox realization within 80 seconds.

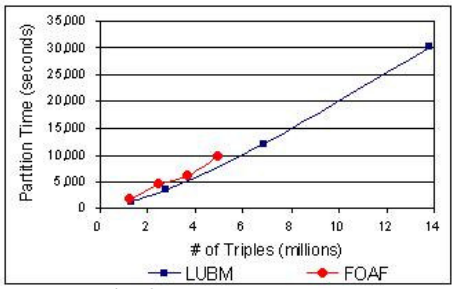

Fig. 3. Partition Time

In order to see how our approach works with real world data, we have also conducted a test on some FOAF data. Thanks to Tim Finin of UMBC, who provided us access to SWOOGLE's index of urls, we have collected a large amount of FOAF data from the Web. The sizes of data we used range from 1M to 5M triples, and for those smaller data sets, they are randomly selected subsets of the largest one.

[2] Using SWOOGLE, we estimate that the Web contained about 45M RDF/OWL triples as of Nov. 2005.

The most interesting OWL features with respect to partitioning in the FOAF ontology are *FunctionalProperty* and *InverseFunctionalProperty* (which translate into number restrictions in DL). For example, when two individuals are stated as having the same *homepage* url, we should infer that they are the same individual, since *homepage* is defined as an *InverseFunctionalProperty* in the FOAF ontology. Our partitioning algorithm has captured a number of such cases according to ChunkRule 3.

Table 2. Test results on FOAF data

Test Set (# of triples)	Partition Time (hh:mm:ss)	# of Partitions	Average Partition Size (# of triples)	Min Partition Size (# of triples)	Max Partition Size (# of triples)
1,240 K	00:29:34	1,117,955	1.4	1	831
2,474 K	01:16:44	1,919,804	1.7	1	1701
3,713 K	01:41:18	2,530,110	2.1	1	2790
4,950 K	02:43:06	2,990,069	2.6	1	4608

Test results on FOAF are shown in Table 2 and Fig. 3. As with the LUBM test, the system scales well. Also the average partition sizes and max partition sizes are very small.[3] This is more than sufficient to partition data sets for main memory reasoning. As a result, our choice to avoid full TBox reasoning is justified.

In order to empirically test the correctness of our implementation, we have done another experiment using a small LUBM dataset (about 8K triples) and a small FOAF dataset (about 5K triples) respectively. For each dataset, we partitioned it and then issued a number of queries on it. For each query, we used Pellet to find answers for each partition and then took the union of them. The test has shown that the answers obtained in this way are identical to the results on the original dataset. Due to lack of space, we omit the details of the test.

Again, we emphasize that the primary goal of this work is to help existing reasoning systems overcome memory limitations in order to handle larger data sizes. Nevertheless, we also consider it would be interesting to investigate how partitioning will impact reasoning and query efficiency, especially in the case where the ABox is big but still can fit into memory. This will be our future work.

5 Related Work

Grau et. al. [5] extended OWL to a family of ε-Connection languages for defining and instantiating combinations of OWL knowledge bases. Based on that, they provided a technique for decomposing an OWL knowledge base into smaller, connected knowledge bases. Amir and McIlraith [1] have researched partition-based reasoning for first-order and propositional theories. Their framework includes the algorithms for automatic partitioning of a theory and the message passing algorithms for reasoning with the partitioned theory. The fundamental difference of the above work from ours lies in that their approaches, due to different requirements and goals, do not guarantee an independent partitioning according to Definition 2. This is because both approaches generate partitions with potential links and correct reasoning may require communication between partitions through the links. In addition, to the best of our knowledge, neither approach has a secondary storage based implementation.

[3] Recall in practice we can merge the partitions in order to reduce their number.

Fokoue et al. [3] have worked on making consistency checking more scalable for ABoxes. The key idea is to employ static analysis of knowledge representation and summarization techniques to extract a reduced proxy ABox, which is consistent only if the original ABox is also consistent. Like our work, they assume large ABoxes in secondary storage. We consider one major difference between the two works is the following: from the instantiation query point of view, the extracted proxy ABox depends on the given query while the partitioning by our approach does not.

Stuckenschmidt and Klein [17] have studied partitioning OWL TBoxes for modeling purposes. Their method automatically partitions a large TBox into smaller components based on the structure of the class hierarchy in that TBox. Unlike our work, their partitioning is not concerned with the influence of the partitioning on the results of reasoning. Also our interest is on the ABox.

As some other related work, in logic programming field, magic set is a well known approach to avoid the consideration of irrelevant facts during the forward inference process. In addition, there has been some research on the technique and language support for parallel execution of logic programs. (See [18] for an example.) The underlying idea of this work is to detect and utilize the dependency between logical variables. Most of the work focuses on Horn Logic programs and languages such as Prolog, and to the best of our knowledge, no similar work has been done with respect to DLs.

6 Conclusion and Future Work

We have contributed an approach for partitioning an OWL knowledge base. Specifically, the approach partitions an OWL ABox (with respect to an TBox) so that we could reason with each partition separately while still guaranteeing sound and complete reasoning about instance checking for concepts and roles. As a result, very large ABoxes can be processed by contemporary main memory-based DL reasoners. Moreover, we want to point out that reasoning with partitions could be done in parallel, and depending on the number of nodes available, the total reasoning time could be as low as x (where x is time to partition + time to reason with the largest partition + time to union results).

In developing our approach, we sacrificed the granularity of partitioning in order to avoid a partitioning procedure that is as complex as DL reasoning. First, in order to quickly determine relevant assertions, we have made some changes to a set of inference rules for OWL Lite, mainly by relaxing the preconditions and consequences of the rules. Then according to the adapted rules, we developed a partitioning algorithm. The algorithm is based on the construction of a chunk graph. Chunks are the building blocks of partitions, and the structure of the chunk graph dictates which chunks must coexist in a partition. In our approach, a lot of decision making depends only on the TBox. We consider this is important for acquiring a scalable partitioning. Overall, our algorithm has worst-case polynomial time complexity.

We have implemented the approach using a database management system and conducted an empirical evaluation using both the Lehigh University Benchmark data and the real world FOAF data. The results have demonstrated good scalability of the approach. Moreover, the small average and max sizes of the resultant partitions have justified the tradeoff we have made between the complexity and the granularity of partitioning. We plan to conduct more experiments in the future to validate this.

For future work, we will continue improve the approach in terms of the granularity of partitioning. Other future work includes extending the technique to OWL DL knowledge bases. In addition, the present work is only concerned with the inference of atomic assertions and we plan to consider conjunctive queries. Lastly, we intend to extend the algorithm to address the case of knowledge base updates. The goal is to avoid having to re-partition from scratch when updates occur.

Acknowledgement: This material is based upon work supported by the National Science Foundation under Grant No. IIS-0346963.

References

[1] Amir, E. and McIlraith S. Partition-Based Logical Reasoning. In Proc. of KR2000.
[2] Ding, L. et al. Search on the Semantic Web. IEEE Computer 10(38):62–69, 2005.
[3] Fokoue, A. et al. 2006. SHIN ABox Reduction. In Proc. of DL2006.
[4] Gallier, J. Logic for Computer Science: Foundations of Automatic Theorem Proving. Harper & Row Publishers, 1986.
[5] Grau, B.C. et al. Automatic Partitioning of OWL Ontologies Using ε-Connections. In Proc. of DL2005.
[6] Guo, Y. and Heflin, J. On Logical Consequence for Collections of OWL Documents. In Proc. of ISWC2005.
[7] Guo, Y., Pan, Z. and Heflin, J. 2005. LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics 3(2), 2005, pp158-182.
[8] Haarslev, V. and Möller, R. 2003. Racer: A Core Inference Engine for the Semantic Web. In Workshop on Evaluation on Ontology-based Tools, ISWC2003.
[9] Haarslev, V. and Möller, R 2004. Optimization Techniques for Retrieving Resources Described in OWL/RDF Documents: First Results. In Proc. of KR2004.
[10] Haarslev, V., Möller, R. and Wessel, M. 2004. Querying the Semantic Web with Racer + nRQL. In Proc. of the KI-04 Int'l Workshop on Applications of Description Logics.
[11] Horrocks, I. 1998. The FaCT System. In Tableaux'98.
[12] Horrocks, I. et al. The instance store: DL reasoning with large numbers of individuals. In Proc. of DL2004.
[13] Horrocks, I. and Patel-Schneider, P. Reducing OWL entailment to description logic satisfiability. Journal of Web Semantics 1(4), 2004, pp345-357.
[14] Pellet OWL Reasoner. http://www.mindswap.org/2003/pellet/.
[15] Royer, V. and Quantz, J.J. Deriving Inference Rules for Terminological Logics. Logics in AI, In Proc. of JELIA'92.
[16] Royer, V. and Quantz, J.J. Deriving Inference Rules for Description Logics: a Rewriting Approach into Sequent Calculi. KIT REPORT 111, Dec. 1993.
[17] Stuckenschmidt, H. and Klein, M. 2004. Structure-based partitioning of large class hierarchies. In Proc. of ISWC2004.
[18] Ueda, K. I/O Mode Analysis in Concurrent Logic Programming. In Proc. of Theory and Practice of Parallel Programming, 1995.