http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 9 (5): 909-917, 2010
ISSN 1812-5638

A Modular Approach for Reasoning about Large-Scale Description Logic Knowledge-Base

Yuxin Mao
School of Computer and Information Engineering, Zhejiang Gongshang University,
Xuezheng Street No. 18, Hangzhou 310018, Zhejiang, People's Republic of China

Abstract: In this study, we proposed a modular approach for description logic reasoning to meet the on-demand and scalability requirement semantic-based systems. One typical use of description logic knowledge-base is to support reasoning in semantic-based systems. However, including large description logic knowledge-bases in their complete form in applications would imply unnecessarily huge storage and computational requirement. Therefore, we go beyond the use of static description logic knowledge-base by reusing knowledge dynamically. In particular, we refer to the context-specific contents from large-scale description logic knowledge-bases as description logic modules. A tableau algorithm based on the description logic module representation is given to support modular description logic reasoning. In order to solve the semi-deterministic problem of modular reasoning, we propose an expansion reasoning algorithm for preserving consistency. We also analyzed the time complexity of the modular reasoning algorithm under different conditions. The proposed algorithm improved the performance of description logic reasoning by modulization, especially when the scale of the knowledge-base is very large.

Key words: Modular reasoning, module, modularization, semantic web, tableau algorithm

INTRODUCTION

Recently, as a kind of knowledge representation system, Description Logics (DLs) (Brachman and Schmolze, 1985; Baader and Nutt, 2003) has become popular, with the emergence of the semantic web (Berners-Lee *et al*., 2001; Ilyas *et al*., 2004) and the research efforts of ontologies (Gruber, 1993; Van Heijst *et al*., 1997; Ding and Sun, 2009). Knowledge representation systems based on DLs provide users with various inference capabilities that deduce implicit knowledge from the explicitly represented knowledge. The study of suitable methods for solving the problems of reasoning in DLs has been developed starting with severe restrictions on the expressiveness of the language and on the form of the knowledge base. The original algorithm by Schmidt-Schauβ and Smolka (1991) for ALC as well as subsequent algorithms for more expressive DLs, could be seen as specializations of the tableau calculus for first-order predicate logic (Baader and Sattler, 2001). Tableau algorithms have turned out to be quite useful. They are complete and often of optimal complexity. The tableau-based approach has been developed and used as a mainstream solution to the reasoning problem in DLs.

However, including large-scale DL knowledge-bases in their complete form could incur an unnecessarily high storage and maintenance cost to semantic-based systems, especially when the capacity of a system is limited (e.g., an embedded system). As the size of a DL knowledge-base increases, the performance of the associated semantic-based systems will be quickly degraded as frequent search of knowledge is typically required. Therefore, it is important and urgent to share and reuse large-scale DL knowledge-bases to support semantic-based systems in a more efficient way.

By assuming that semantic-based systems typically need not the complete DL knowledge-base but just a portion of it, we propose to incorporate into systems the capability of knowledge-base modularization. In this study, we propose a modular approach of dynamic reasoning on large-scale DL knowledge-bases. We introduce the concept of DL module as well as the module cache for DL knowledge-base modularization. We present several modular reasoning algorithms based on the module representation. The proposed algorithm improves the performance of DL reasoning by modulization, especially when the scale of the knowledge-base is very large.

DL KNOWLEDGE-BASE MODULARIZATION

Here, we illustrated how to represent and manipulate modules formally based on the semantic structure of DL knowledge-base.

Definition: Generally, a DL knowledge-base K is a pair <T, A>, where T and A are its TBox and Abox. The TBox introduces the terminology like the vocabulary of an

application domain while the ABox contains assertions about named individuals in terms of the vocabulary. The activities of many semantic-based systems rely only on localized information and knowledge as most applications have their specific contexts that they focus on (Ghidini and Giunchiglia, 2001). Our conjecture is that the activities of semantic-based systems need only specific aspects of a large-scale DL knowledge-base. The implication is that there is a need to provide semantic-based systems with the ability to modularize a large-scale DL knowledge-base according to requirements. Taking into account the locality of knowledge reference, we propose to represent those context-specific contents from a large-scale DL knowledge-base as DL modules (Mao *et al*., 2008). We give a formal definition of DL module to facilitate the discussion in the sequel as follows:

Definition 1 (DL Module): Given a DL knowledge-base K = <T, A>, a DL module (or module for short), M, is represented as a tuple <M_c, M_t, M_a, M_k>, where $M_t \subseteq T$, $M_a \subseteq A$, M_c = {DL concept c_i}, where $\forall c_i \in T$.

M_k is a reference to the DL knowledge-base K, which is also called the source of module. M_c contains some concepts from K and is called the concept set of the module. Given M_c, one can populate a corresponding module by searching the DL knowledge-base K for related concepts and storing them as M_t and M_a <M_t, M_a> is called the knowledge set of the module and denotes a local knowledge-base. Note that M_a is a set of individuals from the ABox of the source DL knowledge-base, which makes the module be complete from the point of DL knowledge-base.

Given a DL module M = <M_c, M_t, M_a, M_k>, it has the following features:

- **Completeness:** Although M is a module from the source DL knowledge-base, it has a relatively complete local DL knowledge-base <M_t, M_a>
- **Connectivity:** The concepts in M_t are connected. To a concept c_i, in M_t, there is a relation between c_i and c_j, where c_j is another concept in M_t

Module manipulation: In order to facilitate manipulating modules from large-scale DL knowledge-bases, we also define a set of basic module operations as follows (Table 1):

We can manipulate modules from a large-scale DL knowledge-base efficiently with these operations. In a knowledge reuse framework, the operations can be packed as services, which are invoked by other semantic-based systems.

Table 1: The Operations for Manipulating Modules.

Operator	Input	Output
Extract	A DL knowledge-base K A concept set D	A module M
Store	A module M A module repository R	A boolean value ret
Compare	A DL knowledge-base K Two modules	A degree of discrepancy diff
Retrieve	A concept set D A module repository R	A module M
Merge	A DL knowledge-base K Two modules	A module M

Cache-based knowledge reuse: Compared with DL module, DL knowledge-base can be treated as static and invariant resource to semantic-based systems. Changes of DL knowledge-base require owner privilege and domain experts' intervention, so it is unpractical to let semantic-based systems directly modify and update source DL knowledge-base. However, different systems can have their local repositories of modules from DL knowledge-base. Based on the module representation for DL knowledge-base, we propose to organize the modules of a semantic-based system in a local repository called module cache for dynamic knowledge reuse.

Definition 2 (Module cache): Given a DL knowledge-base K = <T, A>, a module cache, R, is represented as a triple <I, M_t cv>, where M is a module from K, I is the index of M in R and cv is its cache value.

The concept of module cache draws inspiration from the memory caching mechanism. First emerging from data processing, caches work well because of a principle known as locality of reference. A module cache refers to a local repository for retrieved modules of DL knowledge-base for future reuse. Taking modules as cachable objects, module cache stores modules as cache blocks for knowledge reuse.

Caching the modules from a large-scale DL knowledge-base, which are frequently used in semantic-based systems, can improve the performance of semantic-based activities. When we want to use DL knowledge in applications, we first look for a module in a module cache providing formal semantics rather than accessing the source DL knowledge-base directly. For example, when a semantic-based system lacks the knowledge to perform a reasoning task, it needs to access the related DL knowledge-base. If the system has ever extracted the required modules and kept the most active ones in its module cache, the overall speed for reasoning is then optimized.

MODULAR DL REASONING

Reasoning for large-scale dl knowledge-base: One typical use of DL knowledge-base is to support reasoning

of semantic-based systems (Donini and Lenzerini, 1996). Given a DL knowledge-base K = <T, A>, there are four major reasoning tasks (Baader and Nutt, 2003).

- **Satisfiability:** A concept C is satisfiable with respect to T if there exists a model I of T such that C^I is nonempty.
- **Subsumption:** A concept C is subsumed by a concept D with respect to T if $C^I \subseteq D^I$ for every model I of T
- **Equivalence:** Two concepts C and D are equivalent with respect to T if $C^I = D^I$ for every model I of T
- **Disjointness:** Two concepts C and D are equivalent with respect to T if $C^I \cap D^I = \varnothing$ for every model I of T

If, in addition to intersection, a DL system allows one to form the negation of a description, one can reduce other reasoning tasks to the satisfiability problem (Smolka, 1988; Schaerf, 1994). Therefore, we only consider the satisfiability problem and discuss the algorithm for satisfiability in this study.

Given an ALC-concept description, the tableau algorithm for satisfiability tries to construct a finite interpretation that satisfies the concept description. The original approach by Schmidt-Schaulβ and Smolka (1991) and many other studies on tableau algorithm for DLs, introduce the notion of a constraint system for this purpose. A satisfiability algorithm for ALC is presented as follows, based on the constraint systems (Schmidt-Schaulβ and Smolka, 1991) and the tableau algorithm for ALCN proposed by Buchheit *et al*. (1993). The transforming rules of the algorithm are shown in Table 2.

Given a concept description C_0, we can construct an initial constraint system S_0. Starting with S_0, we thus, obtain a set of complete constraint systems $\{S_1, S_2, \ldots, S_k\}$ after a finite number of rule applications. A constraint system S_i is called complete iff none of the transformation rules applies to it. Consistency of a set of complete constraint systems can be decided by looking for obvious contradictions, called clashes.

Table 2: The transforming rules of the satisfiability algorithm for ALC

Name	Rule
S →⊓	$\{x:C_1, x:C_2\} \cup S$ if $x:C_1 \sqcap C_2$ in S, and $x:C_1$ and $x:C_2$ are not both in S.
S →⊔	$\{x:D\} \cup S$ if $x:C_1 \sqcup C_2$ is in S, neither $x:C_1$ nor $x:C_2$ is in S, and $D = C_1$ or $D = C_2$.
S →∀	$\{y:C\} \cup S$ if $x:\forall R.C$ in S, <x, y>:R is in S, and y:C is not in S.
S →∃	$\{<x, y>:R, y:C\} \cup S$ if $x:\exists R.C$ is in S, y is a new variable, and there is no z s.t. both <x, z>:R and z:C in S.
S →∀x	$\{s:C\} \cup S$ if $\forall x.x:C$ is in S, s is in S, and s:C is not in S.

Definition 3 (Clash): A constraint system S_i contains a clash iff it has one of the following forms:

- $\{x:\bot\}$
- $\{x:A, x:\neg A\}$, where A is a concept name.
- $\{x:\leq n\, R\} \cup \{<x, y_i>:R \mid 1 \leq i \leq n+1\} \cup \{y_i \neq y_j \mid 1 \leq i \leq j \leq n+1\}$, where, R is a role name

Any constraint system containing a clash is obviously unsatisfiable. If all constraint systems derived from S_0 are unsatisfiable, then C_0 is unsatisfiable. Otherwise, if there is a S_i containing no clash, it is possible to construct a model and C_0 is satisfiable (Buchheit *et al*., 1993).

Although, we are able to solve the reasoning problem of DLs by using the tableau algorithms, there is a gap between the reasoning services needed by large-scale DL knowledge-bases and those provided by the current systems. As a popular method for DL reasoning, tableau algorithm has a reasonable space complexity but with a high time complexity. Indeed, the algorithms based on tree-automata, which are used to prove EXPTIME-completeness, require exponential time even in simple cases. Donini and Massacci (2000) present a tableau calculus for checking the satisfiability of a concept with respect to a ALC-TBox with general axioms and transform it into the first simple tableau-based decision procedure working in single exponential time. The time complexity of the algorithm is $O(2^{cn})$, for a suitable constant c>1, where n is the size of the input concept and the TBox. To a large-scale DL knowledge-base (e.g., a large-scale domain ontology), the size of the TBox is very large (n>10000). In that case, the cost of reasoning will be too high for real-life applications.

Therefore, how to implement efficient reasoning service for a large-scale DL knowledge-base is still a difficult issue to semantic-based systems. The knowledge involved in DL modules can be reused to support reasoning tasks above. In this study, we try to optimize the reasoning process by DL knowledge-base modularization.

Modular reasoning algorithm: In a system with the modularization mechanism, we can perform a DL reasoning task based on a module cache. The modules in a module cache are used as the local knowledge-bases for

reasoning. We propose a modular DL reasoning algorithm based on the module representation. As mentioned before, we mainly consider the satisfiability problem in this study. Therefore, we try to solve the reasoning task of checking concept satisfiability in this algorithm. The reasoning Algorithm 1 is shown as follows:

Algorithm 1: DL reasoning algorithm based on module cache

Input: a concept description D, a module cache R and a DL knowledge-base K.
Output: a boolean value ret.

```
convert D into NNF
M← retrieve(D, R)
if M = null then
//cannot find a related module and perform complete DL reasoning
   ret←DLReasoning (D, K)
   return ret
end if
//perform modular DL reasoning
ret←DLReasoning(D, M)
return ret
```

We illustrate the procedure of the algorithm as follows:

Step 1: Convert a concept description D into negation normal form (NNF)
Step 2: Retrieve a matching module M from the module cache
Step 3: Use DL reasoning algorithm to perform reasoning for D with the module M as the knowledge-base. The function DLReasoning refers to the process of DL reasoning by using a tableau algorithm
Step 4: If no modules can be selected as the knowledge-bases for reasoning, perform a complete reasoning based on the source DL knowledge-base
Step 4: Return the result of the reasoning

In fact, the process of reasoning based on module is almost the same as that of DL knowledge-base. The only difference is the scale of the reasoning knowledge-base. As the size of a module is far less than that of a large-scale DL knowledge-base, the complexity of the modular reasoning is also restricted to an acceptable extent. Therefore we decrease the complexity of DL reasoning by modularization.

To realize the function DLReasoning in Algorithm 1, we present a modular tableau algorithm for DL reasoning. The table algorithm is based on the algorithm mentioned earlier and tries to check the satisfiability of a concept description.

Algorithm 2: Modular tableau algorithm for DL reasoning

Input: a concept description D, a module $M = \langle M_c, M_t, M_a, M_k \rangle$.
Output: a reasoning result ret.

```
convert D into NNF
construct an initial constraint system ←based on D and <Mt, Ma>
repeat
   //apply the transforming rules in Table 2 to reduce the system
   apply transformation rules to a constraint system
   generate new constraint systems
until no more rules are applicable // finish reduction
//evaluate whether or not each branch is terminated by a clash
if every constraint system leads to a clash then
   ret ← false //there are no satisfiable models
else
   ret ← true //there exists a satisfiable model
end if
return ret
```

We illustrate the procedure of the algorithm as follows:

Step 1: Convert the concept description D into negation normal form (NNF)
Step 2: Construct an initial constraint system S based on D and the knowledge-base $\langle M_t, M_a \rangle$ of M
Step 3: Apply the transforming rules in Table 2 to S and reduce it to generate new constraint systems;
Step 4: If no more transforming rules are applicable to the constraint systems, terminate the reduction
Step 5: If every constraint system in the resulted set contains a clash, it means that D is unsatisfiable. Otherwise, D is satisfiable

If we take a module as the knowledge-base for DL reasoning, then the process of modular tableau reasoning is almost the same as that of a DL knowledge-base. For example, there is a DL knowledge-base K_{tcm}, which contains domain knowledge about Traditional Chinese Medicine (TCM). K_{tcm} is a large-scale DL knowledge-base. The basic concepts and roles for this example are shown in Table 3.

In TCM, Chinese Materia Medica contains the natural medicinal materials used in TCM practice such as plants, animals and minerals, also known as herbal medicine. Formula of herbal medicine defines the basic notions and the theory of prescription. Here, glycyrrhiza, sargassum,

Table 3: Some basic concepts and roles as well as their meanings

Name	Semantics	Meaning
D	Concept	Chinese materia medica
F	Concept	Formula of herbal medicine
R	Role	Contains components
D_g	Concept	Legume
D_s	Concept	Gulfweed
F_g	Concept	Formulas that contain glycyrrhiza
F_s	Concept	Formulas that contain sargassum
F_{ga}	Concept	Formulas that contain glycyrrhiza and angelica
F_{st}	Concept	Formulas that contain sargassum and tangerine peel

angelica and tangerine peel are all herbal medicines. Glycyrrhiza is an individual of legume and sargassum is an individual of gulfweed. Therefore, we have the following axioms in K_{tcm}:

$$F \sqsubseteq \forall R.D \quad (1)$$

$$F_g \sqsubseteq (\exists R.D_g \sqcap F) \quad (2)$$

$$F_s \sqsubseteq (\exists R.D_s \sqcap F) \quad (3)$$

$$F_{ga} \sqsubseteq F_g \quad (4)$$

$$F_{st} \sqsubseteq F_s \quad (5)$$

$$D_g \sqsubseteq D \quad (6)$$

$$D_s \sqsubseteq D \quad (7)$$

According to the basic theory of TCM formula, glycyrrhiza and sargassum cannot be used as components in the same formula. Therefore, we have the following axiom in K_{tcm}:

$$F_g \sqcap F_s \sqsubseteq \perp \quad (8)$$

A module cache is used to store DL modules extracted from K_{tcm}. Assume there are three modules $M_1 = <\{F_{ga}, F_{st}, D_g, D_s\}, M_{t1}, \varnothing, K_{tcm}>$£¬$M_2 = <\{F, F_g, F_s\}, M_{t2}, \varnothing, K_{tcm}>$£¬$M_3 = <\{F, D\}, M_{t3}, \varnothing, K_{tcm}>$ in the cache. M_{t1} contains Eq. 1-3, 8. M_{t2} contains Eq. 2-5, 8. M_{t3} contains Eq. 1-7.

When a user forms a DL concept $F_{new} \equiv F_{ga} \sqcap F_{st}$, in order to represent a new kind of formula, it raises the problem of checking whether the concept description $F_{new} \equiv F_{ga} \sqcap F_{st}$ is satisfiable or not. We can use the modular reasoning algorithm to solve the problem.

With Algorithm 1, we first refer to the module cache and get the module M_2 for modular reasoning. We assume F_{new} is satisfiable and then it has one individual f at least. Therefore, we have an initial knowledge-base $KB = \{M_{t2} \cup \{F_{ga} \sqcap F_{st}\}, \{F_{ga} \sqcap F_{st}(f)\}\}$ for reasoning. We can simplify some axioms in KB as follows:

$$\neg F_g \sqcup (\exists R.D_g \sqcap F) \quad (9)$$

$$\neg F_s \sqcup (\exists R.D_s \sqcap F) \quad (10)$$

$$\neg F_{ga} \sqcup F_g \quad (11)$$

$$\neg F_{st} \sqcup F_s \quad (12)$$

$$\neg F_g \sqcup \neg F_s \quad (13)$$

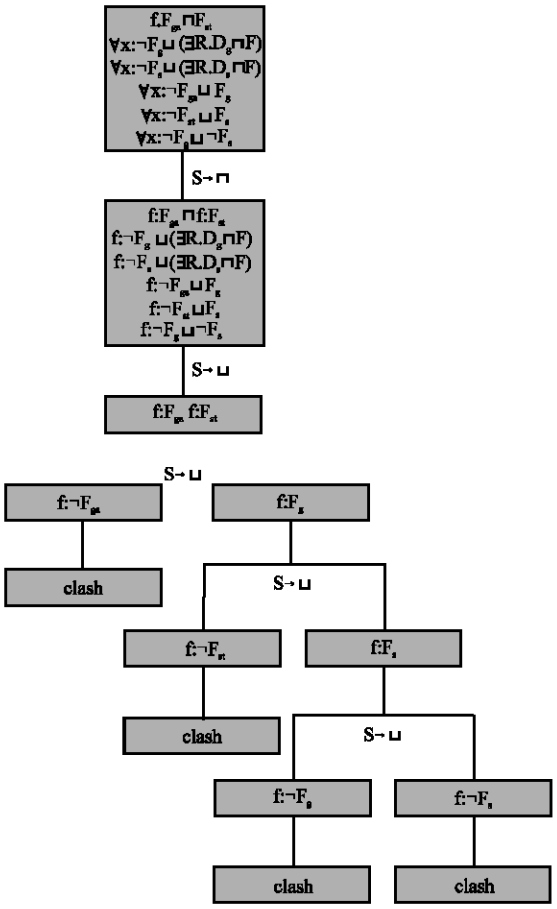

Fig. 1: Example of checking concept satisfiability by using the modular tableau algorithm

Then, we can construct an initial constraint system and perform reasoning according to Algorithm 2. The reduction process is shown in Fig. 1.

When the reduction is finished, we find that all branches are terminated with a clash and it means that every constraint system contains a clash. Therefore, the concept is unsatisfiable and the corresponding formula is a valid one in TCM.

Expansion reasoning: As a module from DL knowledge-base, a module only contains a small portion of knowledge of the complete knowledge-base. Therefore, the knowledge involved in a module is partial compared with its source DL knowledge-base. Assume that the TBox of a DL knowledge-base contains the following knowledge $\{C \sqcup D, \neg C, \forall R.D, E \sqcap F\}$. We extract a module M from the DL knowledge-base including the following knowledge $\{C \sqcup D, R.D\}$. Given a concept description $C \sqcap D$ for checking satisfiability, we first perform modular reasoning based on the knowledge of M and get a

positive answer. However, when we perform reasoning based on the source DL knowledge-base, we just get a clash (¬C and C) and the result is negative. Therefore, the result of modular reasoning is inconsistent with that of complete DL reasoning.

It is trivial to prove that modular reasoning based on module cache is semi-deterministic compared with DL reasoning. In order to keep consistent with the DL reasoning, we should improve Algorithm 1 further. We perform expansion reasoning when we get a positive result by modular reasoning. We propose an expansion reasoning algorithm for this purpose.

Algorithm 3: Expansion reasoning algorithm

Input: a concept description D, a module M, a module cache R and a DL knowledge-base K.
Output: a boolean value ret.

```
ret ← DLReasoning(D, M)
get a set of constraint systems T
if ret = false then
//the result of modular reasoning is negative and return directly
    return ret
else
    //the result of modular reasoning is positive
    ret←false
    //expand each non-conflict branch
    for each constraint system Si without clash in T
      //perform expansion reasoning based on the module cache
      get the concept description Di of Si

    while true
      Mi ← retrieve(Di, R)
      if Mi = null then
        break
      end if
      reti ← DLReasoning(Di, Mi)
      if reti = true then
        ret ← true
        break
      end if
    end loop
    if ret = true then
    break
 end if
 end loop
if ret = false then
//the result of expansion reasoning is negative and return directly
    return ret
    else
//the result of expansion reasoning is positive and perform expansion reasoning based the whole DL knowledge-base
      ret←DLReasoning(D, K)
      return ret
    end if
end if
```

We illustrate the procedure of expansion reasoning as follows:

Step 1: To the conflict branches of modular reasoning, it is no need to expand it anymore

Step 2: To each non-conflict branch, retrieve a matching module from the module cache (Fig. 2)

Step 3: Perform modular reasoning based on the module

Here, step 2 and 3 are performed recursively until no matching modules can be selected.

Step 4: When all the expansion reasoning processes are terminated with conflict, terminate the overall reasoning process. Otherwise, expand the reasoning to the whole source DL knowledge-base

Step 5: Return the final result

If we replace the input concept description $F_{new} \equiv F_{ga} \sqcap F_{st}$ with $\exists R.\neg D \sqcap F_{ga}$, then the reduction process is different. We get the module M_2 from the module cache for modular reasoning. The reduction process is shown in Fig. 3.

We get a constraint system without a clash after reduction. It means that $\exists R.\neg D \sqcap F_{ga}$ is satisfiable with respect to M_2. Then we have to perform expansion reasoning according to Algorithm 3. We get the module M_2 from the module cache according to the final constraint system and perform modular reasoning. The reduction process is shown in Fig. 4.

When the reduction is finished, we find that all branches are terminated with a clash and it means that every constraint system contains a clash. Therefore, the concept is unsatisfiable and this result is consistent with that of K_{tcm}.

Although, the modular reasoning algorithm is semi-deterministic and we have to expand to the whole DL knowledge-base in the worst case, the modular reasoning algorithm does improve the performance of reasoning by modulization in some aspects. We will analyze the performance of the reasoning algorithm in detail later.

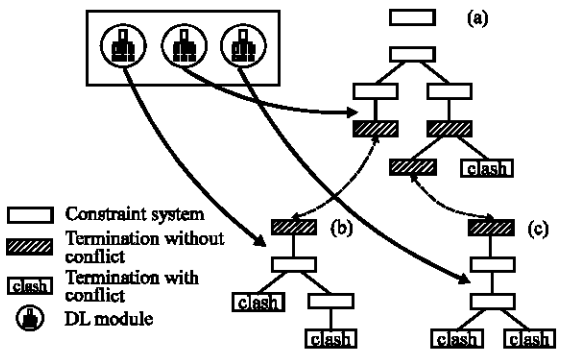

Fig. 2: The expansion reasoning for the modular reasoning algorithm

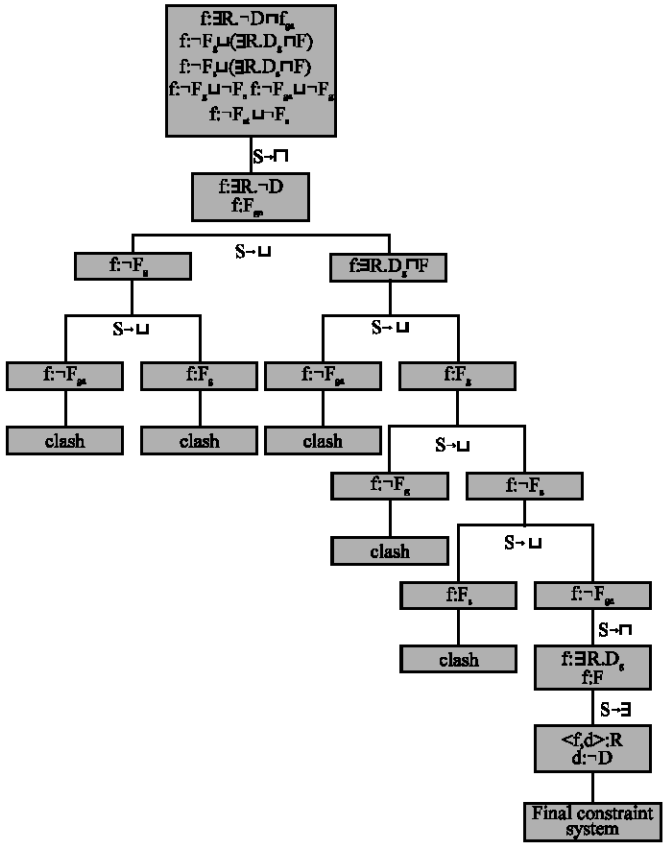

Fig. 3: Example of the expansion reasoning (a)

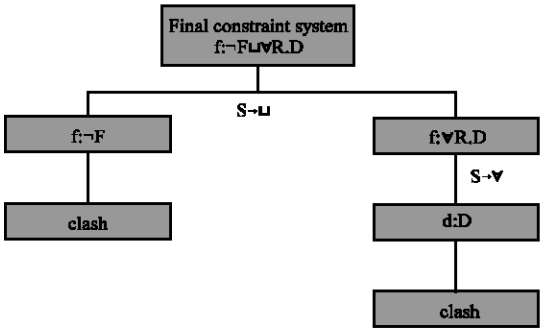

Fig. 4: Example of the expansion reasoning (b)

COMPLEXITY ANALYSIS

Given an ALC knowledge-base K = <T, A> and a DL module M = <M_c, M_t, M_a, K>, the time complexity of the tableau algorithm with K is $O(2^{cn})$, where $n = n_c + n_1$, n_c is the size of the input concept and n_1 is the size of T. The time complexity of the modular tableau algorithm (Algorithm 2) with M is $O(2^{cm})$, where $m = n_c + n_2$, n_2 is the size of M_t. We can analyze the time complexity of the modular reasoning under two different conditions:

- If the result of modular reasoning is negative, the result is consistent with that of complete DL reasoning. We can terminate the reasoning immediately. In this case, the time complexity for modular reasoning is $O(2^{cm})$. As the size of M_t is much smaller than the size of T, it means $n_2 << n_1$ and $O(2^{cn}) << O(2^{cm})$. The time complexity of modular reasoning is much lower than that of DL reasoning in this case
- If the result of modular reasoning is positive, the result is semi-deterministic. In order to keep consistent with the result of complete DL reasoning, we have to perform expansion reasoning. As the expansion procedure is linear, the time complexity is:

$$O\left(\sum_{i=1}^{k} 2^{cn_i}\right)$$

where, n_i is the size of the knowledge set of the ith module in the module cache for expansion reasoning. In the worst case, we have to expand the reasoning to the whole source DL knowledge-base and the time complexity is just $O(2^{cn})$ in that case.

In summary, although the modular reasoning is semi-deterministic, the modular reasoning algorithm could decrease the complexity of reasoning for large-scale DL knowledge-base to some extent. This is mainly because that we reduce the problem space for reasoning by modularization.

DISCUSSION

There have been a number of on-going efforts in modular DL reasoning. Compared with our work, the related works to our approach can be broken down into the following categories: DL knowledge-base or ontology modularity and Modular DL reasoning.

The idea of extracting a subset of a large-scale DL knowledge-base or ontology is referred to by many different names: views, segments, modules, packages, partitions, etc. This research topic is mainly about segmenting or extracting modules from large-scale ontologies to satisfy application requirements. Noy and Musen (2004) defined a portion of an ontology as an ontology view and proposed an approach of specifying ontology views through traversal of concepts. Seidenberg and Rector (2006) proposed some algorithms for extracting relevant segments from large DL ontologies. They interpreted super-classes and quantified restriction as links between classes. Grau *et al.* (2006) defined the notation of locally correct and complete to capture the feature of ontology modularity. They presented an algorithm for automatically identifying and extracting modules from OWL-DL ontologies. Doran (2006) focused on ontology modularisation for reuse. They wanted to separate only part of an ontology, which was correct and self contained and could be reused instead of the original one. It can be concluded from the existing works that reusing large-scale DL knowledge-base by modules can improve the efficiency of semantic-based applications. The DL module presented in this paper is similar with the existing concepts. However, we propose a more formal definition for DL modules that can be reused in a variety of semantic-based applications.

Serafini *et al.* (2005) presented research on knowledge representation and reasoning that support modularity, scalability and distributed reasoning. They indicated a characterization of inferences using a cache-based implementation for local reasoners. Ding and Haarslev (2006) presented a tableau caching technique for SHI to improve the performance of tableau-based DL reasoners. Their results indicate a significant improvement in runtime performance once caching is enabled. Lin (2006) presented a tableau algorithm for concept satisfiability based on ontology modules. If the result is unsatisfiable, the reasoning can be terminated; otherwise, it is still doubtful and the reasoning should go on with the source ontology. Pan (2007) proposed a flexible reasoning architecture for OWL DL and described the prototype implementation of the reasoning architecture based on the FaCT DL reasoner. DL reasoners like RACE also caches concepts that have been reduced or knowledge about unsatisfiablity for future reuse (Horrocks and Schneider, 1999; Haarslev and Möller, 2000). Besides, some research efforts in distributed DLs (Borgida and Serafini, 2002) were concerned with tableau reasoning based on multiple ontology modules (Bao *et al.*, 2006). However, the modules are from different knowledge-bases in that case. From the cases before-mentioned, it can be seen that we can improve DL reasoning by modularity. To contrast with existing research efforts, we propose a modular approach to improve the performance of DL reasoning. We enable semantic-based systems to reuse large-scale DL knowledge-base dynamically by modularization. Our concern of modular reasoning focuses on reasoning about a large-scale DL knowledge-base with a caching mechanism. We also consider the consistent problem between DL modules and the source DL knowledge-base in this study.

CONCLUSION

To semantic-based systems, the ability of reusing large-scale DL knowledge-bases is restricted by the computing and storage capacity of system. In this study, we present a modular approach for reusing large-scale DL knowledge-bases. We define the context-specific contents from large-scale DL knowledge-bases as modules. We combine the caching mechanism with DL knowledge-base reuse to form module cache. Semantic-based systems are able to use module caches as their local knowledge-bases to support semantic-based activities like DL reasoning. We mainly present a modular reasoning algorithm based on the module representation. The algorithm is able to improve the performance of DL reasoning, especially when the scale of the DL knowledge-base is large. There is still much room for improvement on the proposed approach. Future research issues include (1) extending the algorithm to satisfy more DL languages that are more complex than ALC and (2) considering the situation where modules are from different DL knowledge-bases to improve the algorithm.

ACKNOWLEDGMENTS

This work is partially supported by a grant from Educational Commission of Zhejiang Province (No. Y200908082) and a Science and Technology Program of ZJGSU (No. 1130KU110005).

REFERENCES

Baader, A.F. and W. Nutt, 2003. Basic Description Logics. In: The Description Logic Handbook: Theory, Implementation and Applications, Baader, F., D. Calvanese D. McGuinness, D. Nardi and P. Patel-Schneider (Eds.). University Press, New York, ISBN: 0521781760.

Baader, F. and U. Sattler, 2001. An overview of tableau algorithms for description logics. Studia Logica, 69: 5-40.

Bao, J., D. Caragea and V. Honavar, 2006. A tableau-based federated reasoning algorithm for modular ontologies. Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, Dec. 18-22, Hong Kong, China, pp: 404-410.

Berners-Lee, T., J. Hendler and O. Lassila, 2001. The semantic web. Scienti?c Am., 284: 34-43.

Borgida, A. and L. Serafini, 2002. Distributed Description Logics: Directed Domain Correspondences in Federated Information Sources. In: On the Move to Meaningful Internet Systems 2002: CoopIS, DOA and ODBASE. Meersman, R. and Z. Tari (Eds). LNCS., 2519, Springer, Berlin/Heidelberg, ISBN: 978-3-540-00106-5, pp: 36-53.

Brachman, R.J. and J.G. Schmolze, 1985. An overview of the KL-ONE knowledge representation system. Cognitive Sci., 9: 171-216.

Buchheit, M., F.M. Donini and A. Schaerf, 1993. Decidable reasoning in terminological knowledge representation systems. J. Artificial Intell. Res., 1: 109-138.

Ding, B. and L. Sun, 2009. Ontology-based model for software resources interoperability. Inform. Technol. J., 8: 871-878.

Ding, Y. and V. Haarslev, 2006. Tableau caching for description logics with inverse and transitive roles. Proceedings of the International Workshop on Description Logics. http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-189/submission_7.pdf.

Donini, F.M. and M. Lenzerini, 1996. Reasoning in Description Logics: Principles of Knowledge Representation. CSLI Publication, Stanford, CA. USA., pp: 191-236.

Donini, F.M. and F. Massacci, 2000. EXPTIME tableaux for ALC. J. Artificial Intell., 124: 87-138.

Doran, P., 2006. Ontology reuse via ontology modularisation. Proceedings of KnowledgeWeb PhD Symposium, June 17, 2006. Budva, Montenegro. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.9581&rep=rep1&type=pdf.

Ghidini, C. and F. Giunchiglia, 2001. Local models semantics, or con-textual reasoning=locality+ compatibility. Artificial Intell., 127: 221-259.

Grau, B.C., B. Parsia and E. Sirin, 2006. Modularity and web ontologies. Proceeding of the International Conference on the Principles of Knowledge Representation and Reasoning, (CPKPR'06), AAAI Press, pp: 198-209.

Gruber, T., 1993. A translation approach to portable ontology specification. Knowledge Acquisit., 5: 199-220.

Haarslev, V. and R. Möller, 2000. Consistency testing: The race experience. Proceedings of the International Conference on Analytic Tableaux and Related Methods, (ATRM'00), Springer-Verlag, USA., pp: 57-61.

Horrocks, I. and P.F. Schneider, 1999. Optimizing description logics subsumption. J. Logic Comput., 9: 267-293.

Ilyas, Q.M., Y. Zongkai and M.A. Talib, 2004. A journey from information to knowledge: Knowledge representation and reasoning on the web. Inform. Technol. J., 3: 163-167.

Lin, S., 2006. Research on the construction of modular ontology. Ph.D. Thesis, Beijing University of Posts and Telecommunications, China.

Mao, Y., Z. Wu, W. Tian, X. Jiang and W.K. Cheung, 2008. Dynamic sub-ontology evolution for traditional chinese medicine web ontology. J. Biomed. Inform., 41: 790-805.

Noy, N.F. and M.A. Musen, 2004. Specifying ontology views by Traversal. Lecture Notes Comput. Sci., 3298: 713-725.

Pan, J.Z., 2007. A flexible ontology reasoning architecture for the semantic web. IEEE Trans. Knowledge Data Eng., 19: 246-260.

Schaerf, A., 1994. Reasoning with individuals in concept languages. Data Knowledge Eng., 13: 141-176.

Schmidt-Schauβ, M. and G. Smolka, 1991. Attributive concept descriptions with complements. Artificial Intellig., 48: 1-26.

Seidenberg, J. and A. Rector, 2006. Web ontology segmentation: Analysis, classification and use. Proceedings of the International World Wide Web Conference, (IWWW'06), USA., pp: 13-22.

Serafini, L., A. Borgida and A. Tamilin, 2005. Aspects of distributed and modular ontology reasoning. Proceedings of the 19th International Joint Conference on Artificial Intelligence, (IJCAI'05), Edinburgh, Scotland, pp: 570-575.

Smolka, G., 1988. A feature logic with subsorts. Technical Report 33, IWBS, IBM Deutschland, Germany.

Van Heijst, G., A.T. Schreiber and B.J. Wielinga, 1997. Using explicit ontologies in KBS development. Int. J. Human Comput. Stud., 46: 183-292.