

# Scalable Conjunctive Query Evaluation Over Large and Expressive Knowledge Bases

Julian Dolby<sup>1</sup>, Achille Fokoue<sup>1</sup>, Aditya Kalyanpur<sup>1</sup>, Li Ma<sup>2</sup>, Edith Schonberg<sup>1</sup>,  
Kavitha Srinivas<sup>1</sup>, and Xingzhi Sun<sup>2</sup>

<sup>1</sup> IBM Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA  
dolby, achille, adityakal, ediths, ksrinivs@us.ibm.com

<sup>2</sup> IBM China Research Lab, Beijing 100094, China  
malli, sunxingz@cn.ibm.com

**Abstract.** Conjunctive query answering over OWL-DL ontologies is intractable in the worst case, but we present novel techniques which allow for efficient querying of large expressive knowledge bases in secondary storage. In particular, we show that we can effectively answer conjunctive queries without building a full completion forest for a large Abox (unlike state of the art tableau reasoners). Instead we rely on the completion forest of a dramatically reduced summary of the Abox. We demonstrate the effectiveness of this approach in Aboxes with up to 45 million assertions.

## 1 Introduction

Scalable conjunctive query answering is an important requirement for many large-scale Semantic Web applications. Although answering conjunctive queries has been studied from a theoretical point of view, until recently, few reasoners have supported this functionality. To our knowledge, there is no sound and complete conjunctive query answering algorithm that scales to Aboxes with millions of assertions for an expressive description logic such as OWL. In this paper, we provide a solution to this problem. We consider queries with only distinguished variables since such queries are more realistic in practice and can also be answered more efficiently. Our approach is able to handle the expressive DL *SHIN* (OWL-DL minus nominals and datatypes), and scales to Aboxes with up to 45 million assertions.

In our previous work, we have developed an efficient summarization based technique to do sound and complete membership query answering over *SHIN* KBs containing millions of assertions [1]. At its core, the technique applies a standard tableau algorithm to a summary Abox  $\mathcal{A}'$  rather than the original Abox  $\mathcal{A}$ . The summary  $\mathcal{A}'$  is created by aggregating individuals with the same concept sets into a single summary individual. Consistency checking is then performed on  $\mathcal{A}'$ . If the summary is consistent when a negated query is added to a summary individual  $a$ , then all individuals mapped to  $a$  can be ruled out as solutions to the query. If the summary is inconsistent, it is possible that either (i) a subset of individuals mapped to  $a$  are instances of the query or (ii) the

inconsistency is a spurious effect of the summarization. We determine the answer through *refinement*, a process which selectively expands the summary Abox by focusing on inconsistency *justifications* (minimal assertion sets implying the inconsistency), and making them more precise w.r.t the original Abox. Precise justifications are then used to find query solutions. A key point here is that even a precise summary justification is not at the level of Abox individuals, and the scalability of the approach comes from the fact that it makes decisions on groups of individuals as a whole in the summary. For example, when the algorithm concludes that a particular summary individual  $a$  is a solution to the membership query  $C(x)$ , it implies that all the Abox individuals mapped to  $a$  are solutions.

A natural question is whether this technique can be extended to solving relationship queries, and hence arbitrary conjunctive queries. Unfortunately, the nature of our summarization algorithm is such that what works well for solving membership queries does not hold fully for relationship queries. A simple example illustrates the fundamental limitation. Consider the relationship query  $R(x, y)$  over an Abox  $\mathcal{A}$  with summary  $\mathcal{A}'$ . This translates into a membership query over the summary as follows: We add a new atomic concept  $N_y$  as a type to every summary individual in  $\mathcal{A}'$ , and then check for the membership query:  $\exists R.N_y$ . Unfortunately, while this technique gives us all Abox individuals mapped to  $x$ , it cannot tell us all the solution tuples for  $R(x, y)$  in the Abox, because not all Abox justifications for these tuples necessarily appear in the summary in its original structure (for details, see Section 3). As a result, we are forced to find all solution justifications in the Abox, which is an NP-complete problem [2], making this approach impractical.

Our basic approach therefore is to split the conjunctive query into its component membership atoms and relationship atoms, solve these two parts separately, and join the respective bindings at the end. For example, consider a conjunctive query  $C(x) \wedge R(x, y) \wedge D(y) \wedge S(y, z)$  where  $x, y, z$  are distinguished variables,  $C, D$  are concepts and  $R, S$  are roles. Our algorithm does the following:

1. We evaluate the membership queries  $C(x), D(y)$  using an extension of our previous summarization and refinement technique (see section 5), to find potential solution bindings for  $x, y$  (here, variable  $z$  does not have any type constraints and so all Abox individuals are considered potential bindings for  $z$ ). A key point here is to filter out membership query bindings that do not satisfy the remaining relationship constraints in the query, and for this we use the completion forest of the summary Abox to estimate query candidates. This narrows the candidate search space significantly. We prove correctness of this novel optimization in Section 4 (Theorem 1).
2. We then evaluate the relationship query  $R(x, y)$  (resp.  $S(y, z)$ ) by focusing on  $x, y$  (resp.  $y, z$ ) individual bindings found in step 1. Details of how we evaluate relationship queries efficiently, using a Datalog rule-set and optimizations based on the completion forest of the summary Abox, are in Section 6.
3. We join the resultant bindings found across all the relationship query atoms in step 2 to obtain conjunctive query solutions. The entire conjunctive query algorithm that combines steps 1 and 2 is presented in Section 7.

Our contributions in this paper are as follows: (a) we present a technique to perform scalable conjunctive query answering over large and expressive ABoxes which relies on an important new property – using the completion forest of the summary ABox for various optimizations (b) we demonstrate the effectiveness of this technique with very large ABoxes on the UOBM benchmark, (c) we demonstrate graceful degradation of our algorithm’s performance, such that queries whose solutions do not exploit non-determinism in the KB (e.g., do not require non-deterministic mergers between individuals) are performed very efficiently.

## 2 Background

The techniques describe in this paper apply to *SHIN* [3] description logic, which is essentially OWL DL without nominal and datatype.

### 2.1 Definition of Conjunctive Query

Given a knowledge base (KB)  $\mathcal{K}$  and a set of variables  $V$  disjoint with the set  $Ind$  of named individuals in  $\mathcal{K}$ , a conjunctive query  $Q$  is of the form  $(x_1, \dots, x_n) \leftarrow q_1 \wedge \dots \wedge q_m$  where, for  $1 \leq i \leq n$ ,  $x_i \in V$  and, for  $1 \leq j \leq m$ ,  $q_j$  is a query term. A query term  $q$  is of the form  $C(x)$  or  $R(x, y)$  where  $x$  and  $y$  are either variables of named individual in  $\mathcal{K}$ , where  $C$  is a concept and  $R$  is a role both defined in  $\mathcal{K}$ .  $Var(Q)$  refers to the set of variable occurring in query  $Q$ . Let  $\pi : Var(Q) \rightarrow Ind$  be a total function from variables in  $Q$  to named individual in  $\mathcal{K}$ . For a query term  $q$ ,  $\pi.q$  denotes the query term obtained by substituting in  $q$  all occurrences of a variable  $x$  by  $\pi(x)$ .

$(a_1, \dots, a_n)$  is a solution in the KB  $\mathcal{K}$  of the conjunctive query  $Q$  of the form  $(x_1, \dots, x_n) \leftarrow q_1 \wedge \dots \wedge q_m$  iff there is a total function  $\pi : Var(Q) \rightarrow Ind$  such that the following hold : (1), for  $1 \leq i \leq n$ ,  $\pi(x_i) = a_i$ , and (2), for  $1 \leq j \leq m$ ,  $\mathcal{K} \models \pi.q_j$  (i.e.  $\mathcal{K}$  entails  $\pi.q_j$ )<sup>3</sup>.

### 2.2 Summarization and Refinement

In our earlier work, we presented an algorithm based on summarization and refinement to scale consistency check and membership query answering to large ABoxes in secondary storage. A key feature of our algorithm is that we perform consistency detection on a summarized version of the ABox rather than the ABox in secondary storage [4]. A summary ABox  $\mathcal{A}'$  can be constructed by mapping all individuals in the original ABox  $\mathcal{A}$  with the same concept set to a single individual in the summary  $\mathcal{A}'$ . Formally, an ABox  $\mathcal{A}'$  is a summary ABox of a *SHIN*<sup>4</sup> ABox  $\mathcal{A}$  if there is a mapping function  $\mathbf{f}$  that satisfies the following constraints:

<sup>3</sup> Although, per our definition of conjunctive queries, we only discuss in this paper queries with distinguished variables, we can easily extend our approach to support non-distinguished variables that are not involved in any cycle in the query (e.g. by using roll up techniques

<sup>4</sup> We assume without loss of generality that  $\mathcal{A}$  does not contain an assertion of the form  $a \doteq b$

- (1) if  $a : C \in \mathcal{A}$  then  $\mathbf{f}(a) : C \in \mathcal{A}'$
- (2) if  $R(a, b) \in \mathcal{A}$  then  $R(\mathbf{f}(a), \mathbf{f}(b)) \in \mathcal{A}'$
- (3) if  $a \neq b \in \mathcal{A}$  then  $\mathbf{f}(a) \neq \mathbf{f}(b) \in \mathcal{A}'$

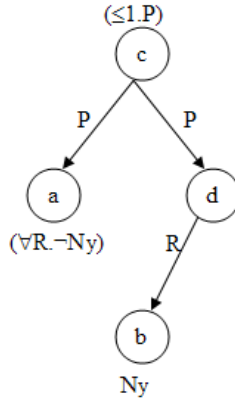
If the summary Abox  $\mathcal{A}'$  obtained by applying the mapping function  $\mathbf{f}$  to  $\mathcal{A}$  is consistent w.r.t. a given Tbox  $\mathcal{T}$  and a Rbox  $\mathcal{R}$ , then  $\mathcal{A}$  is consistent w.r.t.  $\mathcal{T}$  and  $\mathcal{R}$ . However, the converse does not hold. In general, an inconsistency in the summary may reflect either a real inconsistency in the original Abox, or could simply be an artifact of the summarization process.

In the case of an inconsistent summary, we use a process of iterative refinement described in [1] to make the summary more precise, to the point where we can conclude that an inconsistent summary  $\mathcal{A}'$  reflects a real inconsistency in the actual Abox  $\mathcal{A}$ . Refinement is a process by which only the part of the summary that gives rise to the inconsistency is made more precise, while preserving the summary Abox properties(1)-(3). To pinpoint the portion of the summary that gives rise to the inconsistency, we focus on the *justification* for the inconsistency, where a justification is a minimal set of assertions which, when taken together, imply a logical contradiction.

### 3 Why is Solving Relationship Queries using Summarization Hard?

Previously, we have developed an efficient summarization based technique to solve membership queries over very large and expressive KBs [1]. A natural question is whether this technique can be applied (or easily extended) to solving relationship queries. The motivation here is the commonality between the problems, since both membership and relationship query answering are traditionally reduced to KB consistency checking. However, as we shall describe in this section, the nature of our summarization algorithm is such that what works well for solving membership queries does not hold fully for the relationship case.

To understand the fundamental issue with using our previous summarization technique to solve relationship queries, consider a simple example. Suppose we wish to evaluate the relationship query  $R(x, y)$  over an Abox  $\mathcal{A}$  whose summary is  $\mathcal{A}_s$ . We can translate this into a membership query answering problem over the summary as follows: We add a new atomic concept not defined in the TBox, say  $N_y$ , as a type to every summary individual in  $\mathcal{A}_s$ , and then check for the membership query:  $\exists R.N_y$ . Solving this membership query amounts to adding the negation of the queried concept, which is  $\forall R.\neg N_y$ , to every summary individual and checking for newly generated inconsistencies. As described earlier, if the summary is found to be inconsistent, we compute a justification for the inconsistency and continue to refine the portion of the summary corresponding to the justification, until either the summary becomes consistent or the justification becomes precise. Given a precise inconsistency justification  $J$ , we can obtain a relationship query solution as follows: the individual in  $J$  with the type  $\{\forall R.\neg N_y\}$  is the  $x$  solution, while the individual with the type  $\{N_y\}$  is the  $y$  solution.



**Fig. 1.** A precise summary justification  $J$  for the tested individual pair  $a, b$ , where  $a$  has the negation of the query as its type  $-\forall R.\neg N_y$  and  $b$  has the new atomic type  $N_y$ .

For example, Figure 1 depicts a particular precise justification  $J$ , from which we can conclude the solution  $\{x = a; y = b\}$  to  $R(x, y)$ . However, note that the solution pair is in terms of summary individuals, and we need to determine the original Abox individual solution pairs. If we were simply solving the membership query  $\exists R.N_y$ , i.e., we wanted to know all individuals in the Abox that have *some*  $R$  neighbor relation entailed, we could state that all individuals mapped to  $a$  are solutions and we would be done. However, in this case, we are interested in finding specific relationship tuple solutions in the Abox and this is what makes the problem trickier.

Since  $J$  is precise and acyclic, we know that starting with *every* individual mapped to  $a$ , we can reconstruct the entire justification graph pattern for  $J$  in the Abox, leading to *some* individual mapped to  $b$ . Therefore, we can issue an SQL query isomorphic to  $J$ , say  $Q_j$ , against the Abox to obtain solutions to the relationship query. In this case,  $Q_j$  would be ‘select  $x, y$  where  $(\leq 1.P)(v)$  and  $P(v, x)$  and  $P(v, w)$  and  $R(w, y)$ ’, where  $P, R$  are Abox relationships. However, note that  $Q_j$  gives us only one set of relationship tuple solutions in the Abox corresponding to the summary solution  $R(a, b)$  – those whose Abox justification is isomorphic to  $J$ . It is possible that there are individuals mapped to  $a$  which are entailed to have an  $R$  relation to individuals mapped to  $b$  through a separate justification (distinct from  $J$ ). Thus, in order to be complete, we need to find *all* Abox justifications for relationship tuple solutions mapped to summary individuals  $a$  and  $b$  respectively.

There are two drawbacks with this – first, finding all inconsistency justifications in an Abox is an NP-complete problem [2]. Second, and more seriously, attempting to solve this problem at the summary level is incorrect since not all Abox justifications will appear as justifications in the summary. Some Abox

justifications can get masked due to the summarization process. The example in Figure 2 demonstrates this point.

In the sample KB shown in the figure, the Tbox contains one axiom  $\{(C \sqcap D) \sqsubseteq \perp\}$  (i.e.  $C, D$  are disjoint concepts), the Abox graph is shown on the left, and its corresponding non-refinable (precise) summary on the right. The Abox is inconsistent and there are four justifications for the inconsistency as shown:  $J_1 - J_4$ . However, in the corresponding summary, there is only one inconsistency justification  $J_s$ . Even if we issue an SQL query corresponding to  $J_s$  against the Abox, we will obtain two out of the four Abox justifications  $J_1, J_4$ , but we will miss the justifications  $J_2, J_3$  which have been masked due to the summarization algorithm.

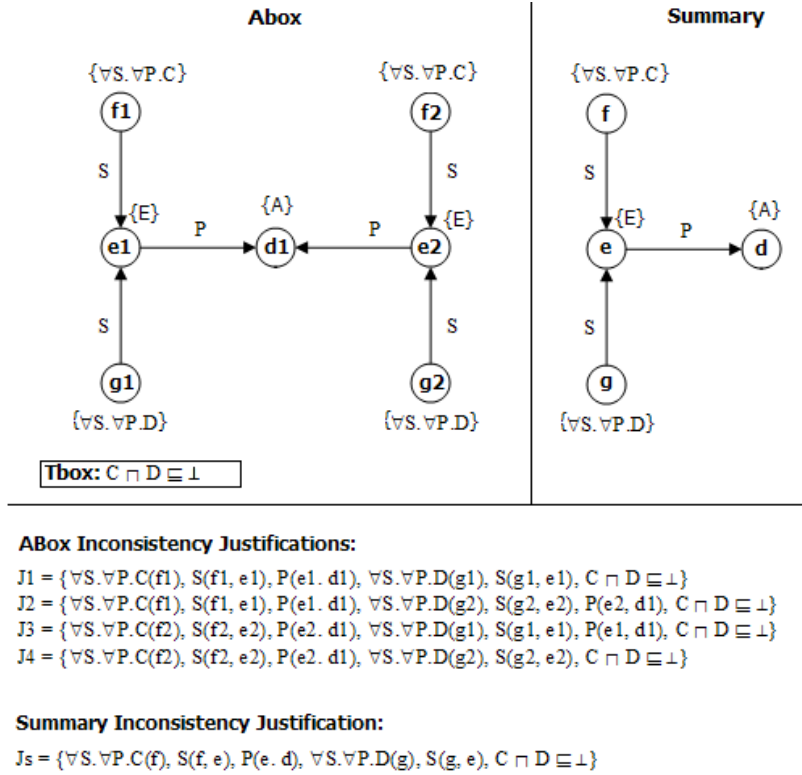


Fig. 2. Abox inconsistency justifications getting masked in the summary

This implies that, in order to do complete relationship query answering using our previous summarization-based technique, we are forced to eventually find all inconsistency justifications in the Abox, which is impractical. Contrast this with

the requirement to find only a single justification in the summary for solving membership queries, and it becomes clear why solving relationship queries using summarization is much harder.

## 4 Optimizing Conjunctive Querying With the Summary Completion Forest

A major limitation to the scalability of conjunctive query evaluation by state-of-the-art tableau-based reasoners stems from their reliance on the completion forest (or pseudo-model) built after application of tableau rules for consistency check. They rely on this completion forest to quickly find obvious solutions, and non-solutions, and therefore efficiently prune out the search space of candidate solutions. Unfortunately, building a completion forest for very large and expressive Aboxes is not possible in practice - both in terms of memory and time required.

In our approach, instead of relying on the completion forest resulting from a direct consistency check on the entire Abox, we use the completion forest obtained from the consistency check on the significantly smaller summary to rule out obvious non-solutions in the Abox. Informally, in order to safely use the completion forest of the summary for pruning purpose, we need to establish the existence of some relationship between it and a completion forest of the original Abox that preserves concept and role sets.

Before describing in more detail the relationship between the summary's completion forest and its Abox's completion forest, we briefly present important tableau algorithm concepts and notations. As described in [3], the tableau algorithm operates on a completion forest  $F = (G, \mathcal{L}, \neq, \doteq)$  where  $G$  is a graph, with nodes corresponding to individuals and edges corresponding to relations;  $\mathcal{L}$  is a mapping from a node  $x$  in  $G$  to a set of concepts,  $\mathcal{L}(x)$ , and from an edge  $\langle x, y \rangle$  in  $G$  to a set of roles,  $\mathcal{L}(\langle x, y \rangle)$ , in  $\mathcal{R}$ ;  $\doteq$  is an equivalence relation corresponding to the equality between nodes of  $G$ ; and  $\neq$  is the binary relation *distinct from* on nodes of  $G$ . At the beginning of the execution of the tableau algorithm on an Abox  $\mathcal{A}$ , the completion forest is initialized as follows: There is a node  $x$  in  $G$  iff there is an individual  $x$  in  $\mathcal{A}$ .  $\langle x, y \rangle$  is an edge in  $G$  with  $R \in \mathcal{L}(\langle x, y \rangle)$  iff  $R(x, y) \in \mathcal{A}$ . For  $x$  and  $y$  in  $G$ ,  $x \neq y$  iff  $x \neq y \in \mathcal{A}$ . Initially, there are no  $x$  and  $y$  in  $G$  such that  $x \doteq y$ . The tableau algorithm consists of executing a set of non-deterministic rules to satisfy constraints in  $\mathcal{A}$ . As soon as an obvious inconsistency, a clash, is detected, the algorithm either backtracks and selects a different non-deterministic choice or stops if all non-deterministic choices have already been made. A *root* node  $a$  is a node present in the initial completion forest (it corresponds to the named individual with the same name in  $\mathcal{A}$ ).

For a root node  $c$  in the completion forest  $F$ , the root node  $\alpha(c)$  is defined as follows (informally,  $\alpha(c)$  corresponds to the node in which  $c$  has been directly

or indirectly merged):

$$\alpha(c) = \begin{cases} c & \text{if } \mathcal{L}(c) \neq \emptyset \\ d & \text{if } \mathcal{L}(c) = \emptyset, d \text{ is the unique root node in } F \\ & \text{with } \mathcal{L}(d) \neq \emptyset \text{ and } d \dot{=} c \end{cases}$$

Now, assuming that the original Abox  $\mathcal{A}$  is consistent, the justifications for the use of the completion forest of its summary for candidate pruning purpose are twofold:

- First, since  $\mathcal{A}$  is consistent, as discussed in [1], a consistent summary  $\mathcal{A}'$  of  $\mathcal{A}$  can always be built through a finite number of refinements.
- Second, as established by theorem 1 below, if  $F'$  denotes the clash-free completion forest resulting from the consistency on the summary  $\mathcal{A}'$  of  $\mathcal{A}$ , then there exists a clash-free completion forest  $F$  resulting from a direct application of tableau rules on  $\mathcal{A}$  such that for two named individuals in  $\mathcal{A}$   $a$  and  $b$  if  $\alpha(\mathbf{f}(b))$  is not a  $R$ -neighbor of  $\alpha(\mathbf{f}(a))$  then  $b$  is not a  $R$ -neighbor of  $a$ . In other words, we can rule out the existence of  $R$ -neighbors of  $a$  in  $F$  based on the non-existence of  $R$ -neighbors of  $\alpha(\mathbf{f}(a))$  in  $F'$ . Therefore, candidate solutions for a query of the form  $R(x, y)$  can be pruned based on completion forest checking on  $F'$  instead of  $F$ .

**Theorem 1** *Let  $K = (\mathcal{A}, \mathcal{T}, \mathcal{R})$  be a consistent knowledge base. Let  $\mathbf{f}$  be a summary mapping function that maps  $\mathcal{A}$  to a consistent summary Abox  $\mathcal{A}'$ . Let  $F'$  be the complete and clash-free completion forest resulting from a consistency check on  $\mathcal{A}'$ ,  $\mathcal{T}$  and  $\mathcal{R}$ . There exists a complete and clash-free completion forest  $F$  resulting from an application of tableau rules directly on  $\mathcal{A}$  such that, for named individuals  $a$  and  $b$  in  $F$  originally present in  $\mathcal{A}$ ,*

- (1)  $\mathcal{L}(a) \subseteq \mathcal{L}'(\alpha(\mathbf{f}(a)))$  (where  $\mathcal{L}(a)$  denotes the concept set of  $a$  in  $F$ , and  $\mathcal{L}'(\alpha(\mathbf{f}(a)))$  is the concept set of the  $\alpha(\mathbf{f}(a))$  in  $F'$ )
- (2) if  $b$  is  $S$ -neighbor of  $a$  in  $F$ , then, in  $F'$ ,  $\alpha(\mathbf{f}(b))$  is a  $S$ -neighbor of  $\alpha(\mathbf{f}(a))$ .

*Proof.* The proof relies on the following main ideas:

- First, to make sure that properties (1) and (2) of Theorem 1 hold, we use  $F'$  to guide the execution of non-deterministic rules on  $\mathcal{A}$  (i.e. we make the same choices as in  $F'$ ).
- Second, we maintain, during the execution of the tableau algorithm on  $\mathcal{A}$ , a mapping  $\sigma$  that maps nodes  $x$  in the completion forest  $F$  obtained from  $\mathcal{A}$  to nodes in  $F'$ , regardless of whether  $x$  refers to a root node, or to a generated node. Furthermore, the relationship between a node  $x$  in  $F$  and  $\sigma(x)$  should be compatible with properties (1) and (2) of Theorem 1. This mapping of  $x$  in  $F$  to nodes in  $F'$  is not straightforward in the presence of blocking, because there is no guarantee that an unblocked generated node  $x$  in  $F$  always maps to a node in  $F'$  that is also not blocked.

We therefore formally define the function  $\sigma$  as mapping a node  $x$  in  $F$  to a pair  $(u, u')$  of nodes in  $F'$ , to handle the case when  $x$  is related to a blocked node  $u'$ . The node  $u$  in the pair is the node that blocks  $u'$  if  $u'$  is blocked; if  $u'$  is not blocked, then  $u$  and  $u'$  are the same ( $u = u'$ ).

Let  $F$  be the completion forest initialized from  $\mathcal{A}$  in the standard way. Before the start of the execution of tableau rules on  $F$ , the function  $\sigma$  maps a root node in  $F$  to a pair of nodes in  $F'$  as follows:

- For a root node  $a$  in  $F$ , we define  $\sigma(a) = (\alpha(\mathbf{f}(a)), \alpha(\mathbf{f}(a)))$

As new generated nodes are introduced during the execution of the tableau rules on  $F$ , the mapping  $\sigma$  is extended to these new nodes as explained below in the treatment of the  $\exists$ -rule and  $\geq$ -rule.  $\sigma(a)[1]$  denotes the first element of the pair  $\sigma(a)$ , and  $\sigma(a)[2]$  is its second element.

We show by induction that at any given step  $k$  of a particular execution <sup>5</sup> of tableau rules on  $F$  the following holds: for all nodes  $x$  and  $y$  in  $F'$

- (A')  $\mathcal{L}_k(x) \subseteq \mathcal{L}'(\sigma(x)[1])$  (where  $\mathcal{L}_k(x)$  denotes the concept set of  $a$  at step  $k$  of the execution of the standard tableau algorithm on  $\mathcal{A}$ , and  $\mathcal{L}'(\sigma(x)[1])$  is the concept set of the  $\sigma(x)[1]$  in  $F'$ )
- (B') if  $y$  is a  $S$ -neighbor of  $x$  and  $y$  is either a root node or a generated child of  $x$ , then, in  $F'$ ,  $\sigma(y)[2]$  is a  $S$ -neighbor of  $\sigma(x)[1]$ .
- (C') for  $\sigma(x) = (u, u')$ ,  $u = u'$  iff.  $u$  is not blocked
- (D') for  $\sigma(x) = (u, u')$ ,  $u \neq u'$  iff.  $u'$  is blocked by  $u$ .
- (E') if  $x \neq y$  holds in  $F$ , then  $\sigma(x)[2] \neq \sigma(y)[2]$  holds in  $F'$

It is very important to note that, since  $F'$  is clash-free, if, at any step  $k$ , (A'), (B') and (E') hold, then, at any step  $k$ ,  $F$  is clash-free.

At step  $k = 0$ , (C') and (D') are obviously satisfied by the definition of  $\sigma$  on root nodes of  $F$ . (A') is satisfied at step 0 as a consequence of the fact that (1) for a root node  $a$  of  $F$ , by definition of a summary mapping, if  $a : C \in \mathcal{A}$  then  $\mathbf{f}(a) : C \in \mathcal{A}'$ , and (2) during the tableau algorithm on  $\mathcal{A}'$ , when root nodes are merged, their concept sets are unioned. A similar argument on role set justifies why (B') holds initially (before the start of the application of tableau rules on  $\mathcal{A}$ ). Finally, (E') holds at step 0 because by definition of a summary mapping, if  $a \neq b \in \mathcal{A}$  then  $\mathbf{f}(a) \neq \mathbf{f}(b) \in \mathcal{A}'$  and two nodes asserted to be different in a completion forest are never merged.

Next we assume that the (A') to (E') hold at a step  $k$  of the execution of the tableau algorithm on  $F$ , and we show that they still hold on the modified forest  $F'$  with a possibly extended  $\sigma$  at step  $k + 1$  - provided the non-deterministic rules are executed in a particular way described in more details below.

$\sqcup$ -rule Let us assume that, at step  $k$ , the  $\sqcup$ -rule is applicable on  $C_1 \sqcup C_2 \in \mathcal{L}(x)$ , and that it is applied at step  $k + 1$ . By the induction hypothesis (A'),  $C_1 \sqcup C_2 \in \mathcal{L}'(\sigma(x)[1])$ , and, since  $F'$  is complete,  $C_1$  or  $C_2$  must be in  $\mathcal{L}'(\sigma(x)[1])$ . Let

<sup>5</sup> An execution in which non-deterministic choices are made based on choices made in  $F'$  as explained in the treatment of non-deterministic rules

assume that  $C_1 \in \mathcal{L}'(\sigma(x)[1])$ . The  $\sqcup$ -rule is then applied by choosing to add  $C_1$  to the concept set of  $x$ . Therefore, at step  $k + 1$ ,  $(A')$  still holds.  $(B')$ ,  $(C')$ , and  $(E')$  still trivially hold.

$\exists$ -rule Let us assume that, at step  $k$ , the  $\exists$ -rule is applicable on  $\exists S.C \in \mathcal{L}(x)$ , and that it is applied at step  $k + 1$ . Let  $\sigma(x) = (u, u')$ . A new generated node  $y$  is created at step  $k + 1$  such that  $\mathcal{L}(\langle x, y \rangle) = \{S\}$  and  $\mathcal{L}(y) = \{C\}$ . By the induction hypothesis  $(A')$ ,  $\exists S.C \in \mathcal{L}'(u)$ . Since  $F'$  is complete, there is a  $S$ -neighbor  $v$  of  $u$  such that  $C \in \mathcal{L}'(v)$ . We extend the definition of  $\sigma$  to  $y$  as follows:

$$\sigma(y) = \begin{cases} (v, v) & \text{if } v \text{ is not blocked} \\ (w, v) & \text{if } v \text{ is blocked by } w \end{cases}$$

$(A')$ ,  $(B')$ ,  $(C')$ ,  $(D')$  and  $(E')$  remain trivially satisfied.

$\geq$ -rule Let us assume that, at step  $k$ , the  $\geq$ -rule is applicable on  $\geq nS \in \mathcal{L}(x)$ , and that it is applied at step  $k + 1$ . Let  $\sigma(x) = (u, u')$ .  $n$  new generated nodes  $y_1, \dots, y_n$  are created at step  $k + 1$  such that, for each  $1 \leq i \leq n$ ,  $\mathcal{L}(\langle x, y_i \rangle) = \{S\}$  and, for  $j \neq i$  and  $1 \leq j \leq n$ ,  $y_i \neq y_j$ . By the induction hypothesis  $(A')$ ,  $\geq nS \in \mathcal{L}'(u)$ . Since  $F'$  is complete, there exists  $n$   $S$ -neighbors  $v_1, \dots, v_n$  of  $u$  that are explicitly asserted to be mutually distinct. For each  $i$  such that  $1 \leq i \leq n$ , we extend the definition of  $\sigma$  to  $y_i$  as follows:

$$\sigma(y_i) = \begin{cases} (v_i, v_i) & \text{if } v_i \text{ is not blocked} \\ (w, v_i) & \text{if } v_i \text{ is blocked by } w \end{cases}$$

$(A')$ ,  $(B')$ ,  $(C')$ ,  $(D')$  and  $(E')$  remain trivially satisfied.

$\leq$ -rule Let us assume that, at step  $k$ , (1) a node  $x$  is not indirectly blocked, (2)  $\leq nS \in \mathcal{L}(x)$ , and (3)  $x$  has  $n + 1$   $S$ -neighbors  $y_1, \dots, y_{n+1}$ . Let  $\sigma(x) = (v, v')$ . By induction hypothesis  $(A')$ , (2) implies that  $\leq nS \in \mathcal{L}(v)$   $(2')$ . By induction hypothesis  $(B')$ , (3) implies that, for each  $1 \leq i \leq n + 1$ ,  $\sigma(y_i)[2]$  is a  $S$ -neighbor of  $v$   $(3')$ . Since  $F'$  is complete,  $(2')$  and  $(3')$  imply that there exist  $i$  and  $j$  such that  $1 \leq i < j \leq n + 1$  and  $\sigma(y_i)[2] = \sigma(y_j)[2]$  (i.e.  $v$  cannot have more than  $n$   $S$ -neighbors). Since  $F'$  is clash-free,  $\sigma(y_i)[2]$  and  $\sigma(y_j)[2]$ , which are identical, cannot be asserted to be distinct. Therefore, induction hypothesis  $(E')$  implies that  $y_i \neq y_j$  cannot hold in  $F$ .

The bottom line here is that  $\leq$ -rule (if  $y_i$  or  $y_j$  is a generated node) or  $\leq_r$ -rule (if  $y_i$  and  $y_j$  are not generated nodes) is applicable at step  $k + 1$ . Let us assume that  $y_i$  or  $y_j$  is a generated node (the case where  $y_i$  and  $y_j$  are not generated nodes is similar). We merge, at step  $k + 1$ ,  $y_i$  and  $y_j$ .

$(A')$  still holds at step  $k + 1$  because (1)  $\sigma(y_i)[2] = \sigma(y_j)[2]$ , (2) since pairwise blocking strategy used for SHIN requires a blocked node and its blocking node to have the same concept set,  $\mathcal{L}'(\sigma(y_i)[1]) = \mathcal{L}'(\sigma(y_i)[2])$  and  $\mathcal{L}'(\sigma(y_j)[1]) = \mathcal{L}'(\sigma(y_j)[2])$ , (3) by the induction hypothesis  $(A')$ , at step  $k$ ,  $\mathcal{L}_k(y_i) \subseteq \mathcal{L}'(\sigma(y_i)[1])$  and  $\mathcal{L}_k(y_j) \subseteq \mathcal{L}'(\sigma(y_j)[1])$ , and (4) the concept set after merger is simply the union of concept sets of merged nodes.

$(B')$  and  $(E')$  still hold at step  $k + 1$  because  $\sigma(y_i)[2] = \sigma(y_j)[2]$ .

$(C')$  and  $(D')$  are still obviously satisfied because the definitions of  $\sigma(y_i)$  and  $\sigma(y_j)$  have not changed.

$\leq_r$ -rule Similar to  $\leq$ -rule.

$\sqcap$ -rule Let us assume that, at step  $k$ , the  $\sqcap$ -rule is applicable on  $C_1 \sqcap C_2 \in \mathcal{L}(x)$ , and that it is applied at step  $k + 1$ . By definition of the tableau  $\sqcap$ -rule, at step  $k + 1$ ,  $C_1$  and  $C_2$  are added to  $\mathcal{L}(x)$ . By the induction hypothesis ( $A'$ ),  $C_1 \sqcap C_2 \in \mathcal{L}'(\sigma(x)[1])$ , and, since  $F'$  is complete,  $C_1$  and  $C_2$  must be in  $\mathcal{L}'(\sigma(x)[1])$ . Therefore, at step  $k + 1$ , ( $A'$ ) still holds. ( $B'$ ), ( $C'$ ), and ( $E'$ ) still trivially hold.

$\forall$ -rule Let us assume that, at step  $k$ , the  $\forall$ -rule is applicable on  $\forall S.C \in \mathcal{L}(x)$ , and that it is applied at step  $k + 1$ . Let  $\sigma(x)$  be such that  $\sigma(x) = (u, u')$ . At step  $k$ , there must be a  $S$ -neighbor  $y$  of  $x$  such that  $C \notin \mathcal{L}(y)$ . By the induction hypothesis ( $A'$ ),  $\forall S.C \in \mathcal{L}'(u)$ . Let  $\sigma(y)$  be such that  $\sigma(y) = (v, v')$ .

- Case 1:  $y$  is either a root node or a generated child of  $x$ . By the induction hypothesis ( $B'$ ),  $v'$  is a  $S$ -neighbor of  $u$ . Since  $F'$  is complete, it follows that  $C \in \mathcal{L}'(v') = \mathcal{L}'(v)$ .
- Case 2:  $y$  is neither a root node nor a generated child of  $x$ . This means that  $y$  is the parent of  $x$ .  $x$  is a  $S^-$ -neighbor of  $y$ . By the induction hypothesis ( $B'$ ),  $u'$  is a  $S^-$ -neighbor of  $v$  (i.e.  $v$  is a  $S$ -neighbor of  $u'$ ). Since, by induction hypotheses ( $C'$ ) and ( $D'$ ) and the definition of pairwise blocking,  $\mathcal{L}'(u) = \mathcal{L}'(u')$ , we must have  $\forall S.C \in \mathcal{L}'(u')$ . Since  $F'$  is complete, it follows that  $C \in \mathcal{L}'(v)$ .

In both cases, ( $A'$ ) remains true at step  $k + 1$  when  $C$  is added to  $\mathcal{L}(y)$ .

( $B'$ ), ( $C'$ ), ( $D'$ ) and ( $E'$ ) remain trivially satisfied.

- $\forall_+$ -rule is similar to  $\forall$ -rule;
- unfold and GCI rules are similar to  $\sqcap$ -rule.

## 5 Solving the Membership Query Part

To evaluate all the membership query atoms in the conjunctive query efficiently, we restrict our tests to candidate individuals that conservatively satisfy all the relationship atoms in the conjunctive query by making use of the completion forest of the summary Abox. Note that in general (as described in [5]), the completion forest of an Abox can be used to rule out candidates  $a, b$  to test for a relationship query  $R(x, y)$ . The intuition here is that a completion forest  $F$  represents an abstraction of a model of the Abox, and thus if  $b$  is not an  $R$ -neighbor of  $a$  in  $F$  (and  $R$  is not transitive), the relation  $R(a, b)$  cannot be entailed by the KB. We apply the same principle to the completion forest of the summary Abox, which is possible due to Theorem 1. Also, we extend this idea by checking possible satisfaction of all the relationships in the conjunctive query.

The algorithm SELECT-CANDIDATES-MQ to select test candidates is shown below. Basically, the algorithm transforms the relationship atoms in the original conjunctive query into a SPARQL query  $Q_r$  and issues it over the completion forest of the summary  $F'$ . Solutions to  $Q_r$  give us candidates to test for the membership constraints.

During the transformation, special care is taken for constants appearing in role atoms. Since  $Q_r$  is evaluated on the summary, constants are replaced by the

SELECT-CANDIDATES-MQ( $F'$ ,  $\mathcal{R}$ ,  $f$ ,  $R_j(x_k, x_l)$  ( $1 \leq j \leq n$ ))

**Input:**  $F'$  Completion forest of Summary Abox,  $\mathcal{R}$  Rbox of the original KB,  $f$  Abox  $\mapsto$  Summary mapping function,  $R_j(x_k, x_l)$  Set of role atoms in original conjunctive query

**Output:**  $\tau(x \mapsto i)$  mapping from variables to summary individuals

- (1)  $V_s \leftarrow$  set of all variables in role atoms  $R_j$  ( $1 \leq j \leq n$ )
- (2)  $R_s \leftarrow$  set of all role atoms  $R_j(x_k, x_l)$  ( $1 \leq j \leq n$ )
- (3) For any constant  $c$  in any of the role atoms in  $R_s$ , obtain the summary individual  $s \leftarrow f(c)$ , and replace  $c$  by  $s$
- (4) Create a SPARQL query  $Q_r$  whose SELECT clause is  $V_s$  and whose WHERE clause is  $\bigwedge R_s$ .
- (5) Issue  $Q_r$  over  $F'$  with only Rbox inferencing using  $\mathcal{R}$  to obtain solution mapping  $\tau(x \mapsto i)$
- (6) Remove individual solutions from  $\tau$  which are considered ‘anonymous’ in  $F'$
- (7) Since  $F'$  may contain mergers between individuals in  $\mathcal{A}'$ , expand any individual binding  $i$  in  $\tau$  by its equivalence set ( $sameAs(i)$ )
- (8) **return**  $\tau(x \mapsto i)$

corresponding summary individuals that they are mapped to. Since we assume that all variables in the original conjunctive query are distinguished, we need to consider the variables in role atoms in the select clause of  $Q_r$ . The query is evaluated considering the Rbox  $\mathcal{R}$  of the original KB, as we would like to capture relationships that can be inferred due to sub-property, inverse or transitive axioms in it (Note that the Tbox need not be considered since we do not care about concepts and concept-related axioms at this point). Since  $F'$  is small, evaluating this query is straightforward.

The result of executing  $Q_r$  is a mapping  $\tau$  from variable to summary individuals, the latter becoming test candidates for the membership query constraints on the former. Note that the completion forest of the summary Abox may contain ‘anonymous’ individuals that are generated due to the presence of existential quantifiers in the KB. Obviously, these anonymous summary individuals are not present in the original Abox either and so we do not need to test them. Therefore, we discard any anonymous individuals from  $\tau$ .

Having identified suitable test candidates, we now proceed to test them for their respective membership query atoms, using our summarization and refinement techniques [1]. The algorithm SOLVE-MQ sketched below does this.

While the techniques in [1] focused on testing a single membership query  $C(x)$  on the summary, they can be easily extended to test multiple membership queries  $C_i(x)$  ( $1 \leq i \leq m$ ) on the summary at the same time. The main difference is that we now start by adding the negation of all the membership types  $C_i$  to their respective summary individual candidates, before testing for inconsistency. The rest of the approach remains more or less the same – we continue to find and refine inconsistency-causing justifications till we can conclude on pre-

SOLVE-MQ(  $Q, \mathcal{A}, \mathcal{T}, \mathcal{R}$  )

**Input:**  $Q$  the conjunctive query,  $\mathcal{A}$  Abox,  $\mathcal{T}$  Tbox,  $\mathcal{R}$  Rbox

**Output:**  $\mathcal{A}'_c$  consistent version of summary Abox,  $\mathbf{f}_c$  summary mapping function for  $\mathcal{A}'_c$ ,  $F'_c$  completion forest of  $\mathcal{A}'_c$ ,  $\beta$  mapping from a variable to summary individuals satisfying its type constraints

- (1)  $(\mathcal{A}', \mathbf{f}) \leftarrow$  compute summary abox of  $\mathcal{A}$  and its mapping function  $\mathbf{f}$
- (2)  $(\mathcal{A}'_c, \mathbf{f}_c) \leftarrow$  consistent version of  $\mathcal{A}'$  and its mapping function obtained through refinement
- (3)  $F'_c \leftarrow$  complete and clash-free completion forest of  $\mathcal{A}'_c$
- (4)  $\tau \leftarrow$  SELECT-CANDIDATES-MQ( $F'_c, \mathcal{R}, \mathbf{f}_c, R_j(x_k, x_l) \in Q$ )
- (5) **foreach** variable  $x_k$  in  $Q$
- (6)     **if** variable  $x_k$  has type constraints in  $Q$
- (7)          $\beta(x_k) \leftarrow$  compute, through refinement, summary individuals in  $\tau(x_k)$  instances of concept  $\bigcap_{C_p(x_k) \in Q} C_p$
- (8)     **else**
- (9)          $\beta(x_k) \leftarrow \tau(x_k)$
- (10) **return**  $(\mathcal{A}'_c, \mathbf{f}_c, F'_c, \beta)$

cise summary justifications (or the summary becomes consistent). Looking at a precise justification tells us the tested individual and the corresponding membership query it is a solution for, provided that the justification contains only one tested individual. Having found a solution, we stop testing the concerned individual and continue. We have to be careful about justifications containing multiple tested individuals, since in this case, the inconsistency could be a fake one produced by the interaction between the tested individuals and the newly added negated concepts. Therefore, we do not conclude on such justifications, unless they are the only ones left in the summary, in which case we test each summary individual separately and continue.

It is important to note that the most costly step in the algorithm is that of refinement since it involves issuing a query to the Abox database to determine how to make a portion of the summary more precise. Therefore, we gain a lot by testing all the membership atoms on the summary simultaneously since it allows us to do a single refinement step taking into account all the tested individuals and their respective justifications. The alternative of testing each membership atom on a separate summary means more refinement steps in total, and thus more evaluation time.

In addition, we have developed a new optimization to make membership querying faster that relies on known solution justification patterns. The idea is that at any stage of the algorithm, if we have knowledge about a particular solution pattern (e.g. an acyclic precise justification in the summary is a solution pattern), we issue queries corresponding to the pattern directly against the Abox to obtain new solutions and stop testing summary individuals corresponding to the solutions. Even before we summarize the Abox, we use standard query expansion techniques [6] to obtain sound but incomplete solutions as quickly as

possible, and stop testing these solutions in the summary Abox. Details of our own query expansion algorithm are in [7].

As a sidenote, we have experimented with the option of making the membership queries in the first step more focused on the entire conjunctive query by using the standard query-rolling-up technique. For example, instead of issuing the membership query  $C(x)$ , we can roll-up the entire conjunctive query from the perspective of variable  $x$  to produce the query  $Q_x$ :  $(C \sqcap \exists R.(D \sqcap N_y \sqcap \exists S.N_z)(x)$  where  $N_y, N_z$  are new atomic concepts added to every summary individual, and can be used to find additional  $y, z$  bindings. The objective here is to find more precise bindings for  $x$  by evaluating  $Q_x$  (which encapsulates role constraints from the conjunctive query as well), and then use the axioms in the justification of the solutions to find corresponding  $y, z$  bindings. To illustrate this point, consider a possible justification for the solution  $Q_x(a)$ :  $\{-Q_x(a), R(a, b), D(b), N_y(b), S(b, c), N_z(c)\}$  (note that the standard tableau technique to test  $Q_x$  is to add the negation of the concept and check for inconsistency). Looking at this inconsistency justification for the solution  $(x : a)$ , we can also infer that  $b, c$  are bindings for  $y, z$  respectively. The same rolling-up procedure can be done for the remaining variables in the conjunctive query to find a complete set of precise bindings in this step.

However, the experimental results of this approach were quite poor. The main problem was that the rolled-up membership queries were much more complex (as can be seen for  $Q_x$  above), giving rise to larger and more complex justifications in the summary, slowing down the refinement process, and making this step particularly slow on large conjunctive queries. Another drawback is that rolling-up is only possible for acyclic conjunctive queries. An additional issue is that when we focus only on membership query atoms, we have the possibility of finding ‘obvious’ solutions to the query upfront (i.e. when the type of a summary individual is inferred to be a sub-type of the queried concept, we consider these as obvious summary solutions and do not test them); whereas the rolled-up complex queries do not have any obvious solutions. Finally, we found that the earlier approach of focusing only on membership query atoms and using the summary completion forest to efficiently prune test candidates, not only worked much faster but also wasted little time on non-solutions; and therefore we opted for it.

## 6 Solving the Relationship Query Part

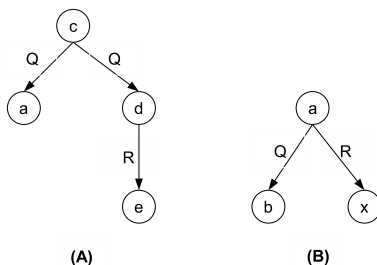
In this section, we discuss how we evaluate each of the role atoms  $R(x, y)$  in the conjunctive query. We solve an atomic role query in three steps:

1. Section 6.1: We estimate an upper bound on potential relationship solutions for  $R(x, y)$  in the Abox by capturing all possible ways in which relationships can be inferred in  $\mathcal{SHIN}$ . We do this efficiently using using the completion forest of the summary Abox and a set of Datalog rules. The rules are restricted to the membership query solutions that are output in the previous step.

2. Section 6.2: After estimating potential role assertion solutions in the Abox, we identify *definite* or deterministically-derived role assertions, since we do not have to test for them.
3. Section 6.3: Finally, we test and solve the remaining potential relationship solutions in the summary Abox.

### 6.1 Estimating Potential Solutions for an Atomic Role Query $R(x, y)$

Our approach to estimate potential solutions to role queries consists in first understanding how, in the completion forest  $F$  of an Abox  $\mathcal{A}$ , a root node can acquire new root node  $R$ -neighbors (i.e. root node  $R$ -neighbors that were not present before the beginning of rule execution). Then, we devise a set of simple rules (see Figure 4) to conservatively estimate potential  $R$ -neighbors. These rules are simple enough to be efficiently evaluated using a datalog engine. Figure 3



**Fig. 3.** Acquisition of named individual  $R$ -neighbors

illustrates the two ways a root node  $a$  in  $F$  can acquire new  $R$ -neighbors that are root nodes during the execution of the tableaux algorithm on  $F$ :

- (A) The root node  $a$  is merged with another root node  $d$  and acquires root node  $R$ -neighbors of  $d$ . The merger is performed to satisfy the maximum cardinality restriction  $\leq nQ$  in the concept set of  $c$ . This case also captures acquisition of  $R$ -neighbors through mergers involving root neighbors of  $a$ .
- (B) The root node  $b$  is merged with a generated node  $x$  to satisfy the maximum cardinality restriction  $\leq nQ$  in the concept set of  $a$ . As a result of this merger,  $b$  becomes a  $R$ -neighbor of  $a$  since  $x$  was a  $R$ -neighbor of  $a$ .

Let us assume that  $F'$  is a complete and clash-free completion forest of the summary  $\mathcal{A}'$  of the Abox  $\mathcal{A}$ , and  $F$  is the complete and clash-free completion forest of  $\mathcal{A}$  given by Theorem 1.

We can conservatively account for acquisition of named  $R$ -neighbors of  $a$  through mergers with named individuals by applying rules (see rules *NamedMerge*, *SameRel1*, and *SameRel2* in Figure 4) on the Abox that trigger a merger between  $a$  and  $d$  if (1)  $a$  is a  $Q$ -neighbor of  $c$  in  $\mathcal{A}$  (explicitly or as a result of

evaluation of our simple rules), (2)  $d$  is a  $Q$ -neighbor of  $c$  in  $\mathcal{A}$  (explicitly or as a result of evaluation of our simple rules), and (3), in the completion forest  $F'$ ,  $\leq nQ \in \mathcal{L}'(\alpha(\mathbf{f}(c)))$ . If the last condition is not satisfied, Theorem 1 guarantees that a merger between  $a$  and  $d$  is not possible in  $F$  since  $\leq nQ$  cannot be the concept set of  $c$  in  $F$ .

One way to account for mergers between root nodes and generated nodes is to have rules that create these generated nodes. However, this is not practical because too many nodes might be generated, complex blocking mechanism will be required to ensure termination, and the resulting rules will not be simple enough to be efficiently evaluated by a datalog engine.

Our approach to conservatively account for mergers illustrated in Figure 3 (B) is to first observe that in order for them to occur in  $F$ , the following conditions must be satisfied:

- a role generator ( $\exists S.C$  or  $\geq mS$ , where  $S$  is a role in the Rbox) must be in the concept set of  $a$  (otherwise,  $a$  cannot have a generated node as its neighbor), and
- a maximum cardinality restriction  $\leq nQ$  must be in the concept set of  $a$  and, the following must hold:
  - $b$  must be a  $Q$ -neighbor of  $a$ , and
  - $x$  must be a  $Q$ -neighbor of  $a$ .

For a named individual  $a$  in the abox  $\mathcal{A}$ , Theorem 1 allows us to check whether a maximum cardinality  $\leq nQ$  and a role generator concept ( $\exists S.C$  or  $\geq mS$ ) can be present in the concept set of  $a$  in the completion forest  $F'$  of  $\mathcal{A}$  simply by checking whether they are in concept set of  $\alpha(\mathbf{f}(a))$  in  $F'$ . This reduces the number of potential individuals  $a$  and  $b$  such that  $b$  can become a  $R$ -neighbor of  $a$  through mergers of type (B). To further reduce this number, we need a good upper bound on the set  $\phi_a$  of roles  $P$  such that there is a generated node  $x$   $P$ -neighbor of  $a$  in  $F$ , since  $R$  has to be in  $\phi_a$ . A direct consequence of property (B') in the proof of Theorem 1 is that the following set is such an upper bound:  $\{P \mid \text{there is a } P\text{-neighbor of } \alpha(\mathbf{f}(a)) \text{ in } F'\}$ .

Let  $\widehat{\phi_{\mathbf{f}(a)}}$  be an upper bound of the set  $\phi_a$  that depends only on information in  $F'$ . We can now express all the necessary conditions for  $b$  to possibly become a  $R$ -neighbor of  $a$  in  $F$  through a merger of type (B) in terms of information present in  $F'$ :

- an existential restriction  $\exists S.C$  or a minimum cardinality restriction  $\geq mS$  must be in the concept set of  $\alpha(\mathbf{f}(a))$  in  $F'$ .
- a maximum cardinality restriction  $\leq nQ$  must be in the concept set of  $\alpha(\mathbf{f}(a))$  and, the following must hold:
  - $b$  must be a  $Q$ -neighbor of  $a$  (either explicitly in  $\mathcal{A}$  or through the application of rules to estimate potential new mergers)
  - $\{Q, R\} \subseteq \widehat{\phi_{\mathbf{f}(a)}}$  (because there must be a generated node  $x$  which is both a  $Q$ -neighbor of  $a$  and a  $R$ -neighbor of  $a$  in  $F$ ).
- finally,  $\alpha(\mathbf{f}(b))$  must be a  $R$ -neighbor of  $\alpha(\mathbf{f}(a))$  (direct consequence of Theorem 1 and the fact that  $b$  has become  $R$ -neighbor of  $a$  in  $F$ )

Based on the previous necessary conditions, rule *UnnamedMerge* in Figure 4 accounts for potential acquisition of new  $R$ -neighbors in  $F$  through merger of type (B).

For transitive roles, we perform the transitive closure over the estimated inferred neighbors (computed by rules in Figure 4). It is important to note that new relations found after the application of the transitive closure cannot cause merger rules to trigger because, in  $\mathcal{SHLN}$ , maximum cardinality restrictions can only be defined on simple roles (i.e. roles which are not transitive and do not have transitive subrole).

Finally, the rule *Relevance* in Figure 4 forces the rule engine to focus only on relationships appearing in the conjunctive query  $Q$ , and on the individual solutions which satisfy the membership constraints in  $Q$ , specified by the mapping  $\beta$  in the output of algorithm SOLVE-MQ.

<i>(Init)</i>	$InfTriple(X, R, Y)$	$:- R(X, Y) \in \mathcal{A}$
<i>(SameSym)</i>	$same(X, Y)$	$:- same(Y, X)$
<i>(SameTrans)</i>	$same(X, Y)$	$:- same(X, Z) \text{ and } same(Z, Y)$
<i>(NamedMerge)</i>	$same(X, Y)$	$:- \mathbf{f}(Z) = A \text{ and } \leq nR \in \mathcal{L}'(\alpha(A)) \text{ and } X \neq Y$ $\text{and } InfTriple(Z, R, X) \text{ and } InfTriple(Z, R, Y)$
<i>(SameRepl1)</i>	$InfTriple(X, R, Y)$	$:- same(X, Z) \text{ and } InfTriple(Z, R, Y)$
<i>(SameRepl2)</i>	$InfTriple(X, R, Y)$	$:- same(Y, Z) \text{ and } InfTriple(X, R, Z)$
<i>(UnnamedMerge)</i>	$InfTriple(X, R, Y)$	$:- \mathbf{f}(X) = A \text{ and } \mathbf{f}(Y) = B \text{ and } \leq nT \in \mathcal{L}'(\alpha(A))$ $\text{and } (\exists S.C \in \mathcal{L}'(\alpha(A)) \text{ or } \geq mS \in \mathcal{L}'(\alpha(A)))$ $\text{and } (\alpha(B) \text{ is a } R\text{-neighbor of } \alpha(A) \text{ in } F')$ $\text{and } \{R, T\} \subseteq \widehat{\phi}_A \text{ and } InfTriple(X, T, Y)$
<i>(SubRole)</i>	$InfTriple(X, R, Y)$	$:- S \sqsubseteq^* R \text{ and } InfTriple(X, S, Y) \text{ and } S \neq R$
<i>(InvRole)</i>	$InfTriple(X, R, Y)$	$:- S^- = R \text{ and } InfTriple(Y, S, X)$
<i>(Relevance)</i>	$RelInfTriple(X, R, Y)$	$:- InfTriple(X, R, Y) \text{ and } \mathbf{f}(X) = A \text{ and } \mathbf{f}(Y) = B$ $\text{and } R(x_1, x_2) \in Q \text{ and } A \in \beta(x_1) \text{ and } B \in \beta(x_2)$ $\text{and } A \text{ is an } R\text{-neighbor of } B \text{ in } F'$

**Fig. 4.** *PotentialRuleSet*: Rules to compute potential new named individual neighbors that are relevant to conjunctive query  $Q$ . Main output: *RelInfTriple*

## 6.2 Finding Definite Role Assertions

After estimating potential role assertion solutions in the Abox, we identify *definite* or deterministically-derived role assertions, since we do not have to test for them.

In particular, consider the rule *NamedMerge* in Figure 4 which conservatively estimates potential mergers between named Abox individuals. We can be more precise here for deterministic mergers if we somehow identify which Abox individuals mapped to summary individual  $A$  are entailed to be of type  $\leq 1.R$ . Conceptually, this amounts to solving the membership query  $\leq 1.R(A)$  in the summary Abox, which we evaluate efficiently using our membership query

answering solution. Similar analysis is done for the rule *UnnamedMerge* to identify Abox individuals that have role-generators ( $\geq m.S$  or  $\exists S.C$ ) as an entailed type. This gives us two new rules – *DefnNamedMerge*, *DefnUnnamedMerge* – shown in Figure 5, which replace the rules *NamedMerge*, *UnnamedMerge* in the *PotentialRuleSet* (Figure 4) to produce the rule set *DefnRuleSet* that computes definite Abox relationship solutions.

$$\begin{array}{ll}
(\text{SummaryKB Defn}) & \mathcal{K}' = (\mathcal{A}', \mathcal{T}, \mathcal{R}) \\
(\text{DefnNamedMerge}) & \text{same}(X, Y) \quad :- \mathbf{f}(Z) = A \text{ and } \mathcal{K}' \models_{\leq} 1R(A) \text{ and } X \neq Y \\
& \quad \text{and } \text{InfTriple}(Z, R, X) \text{ and } \text{InfTriple}(Z, R, Y) \\
(\text{DefnUnnamedMerge}) & \text{InfTriple}(X, R, Y) \quad :- \mathbf{f}(X) = A \text{ and } \mathbf{f}(Y) = B \text{ and } \mathcal{K}' \models_{\leq} 1T(A) \\
& \quad \text{and } (\mathcal{K}' \models \exists S.C(A) \text{ or } \mathcal{K}' \models_{\geq} mS(A)) \\
& \quad \text{and } S \sqsubseteq^* R \text{ and } S \sqsubseteq^* T \text{ and } \text{InfTriple}(X, T, Y)
\end{array}$$

**Fig. 5.** *DefnRuleSet*: Obtained by replacing *NamedMerge* and *UnnamedMerge* in the *PotentialRuleSet* with the rules shown

### 6.3 Solving Remaining Potential Role Assertions

Having found potential role assertions solutions for  $R(x, y)$  in the Abox and identifying the definite ones, we are left with testing the remaining potential solutions.

Suppose the remaining potential tuples to test are  $\{R(u_1, v_1), \dots, R(u_n, v_n)\}$ , where  $u_k, v_k$ , ( $1 \leq k \leq n$ ) are Abox individuals. Instead of testing these tuples in the Abox, we test them in the summary, i.e., for a given tuple  $R(u_k, v_k)$  we identify the summary individuals to which  $u_k, v_k$  are mapped, say  $a_i, b_j$  respectively, and test whether  $R(a_i, b_j)$  is entailed in the summary KB. This test is done by reducing the problem to membership query answering as described in the introduction. However, the limitation here is that when we find a tuple solution  $R(b_i, b_j)$  in the summary (where  $b_i, b_j$  are summary individuals), we cannot compute all Abox relationship solutions from it – all we know is that every individual mapped to  $b_i$  is entailed to have an  $R$ -relation to some individual in  $b_j$  (and vice-versa, every individual mapped to  $b_j$  has an  $R^-$  relation to some individual mapped to  $b_i$ ).

In this case, for the sake of completeness, we are left with no choice other than to split one of the summary individuals down to the level of the Abox individuals mapped to it and test for relationships subsequently. Obviously, we choose to split the summary individual which has less Abox individuals mapped to it, to restrict the size of our summary Abox. Even in this worst case scenario, the performance of the algorithm is not severely affected as only one end of the tuple is split and the grouping of individuals is still preserved at the other end. Also, other than the tested tuples, the rest of the summary remains unchanged (so typically a large part of the Abox is still summarized).

We combine the three steps discussed in this section into an algorithm SOLVERQ that finds all solutions to a relationship query.

SOLVE-RQ( $R(x_i, x_j), \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{A}'_c, \mathbf{f}_c, F'_c, \beta$ )  
**Input:**  $R(x_i, x_j)$  Relationship query,  $\mathcal{A}$  Abox,  $\mathcal{T}$  Tbox,  $\mathcal{R}$  Rbox,  $\mathcal{A}'_c$  Consistent Summary of  $\mathcal{A}$ ,  $\mathbf{f}_c$  Abox  $\mapsto$  Summary mapping function,  $F'_c$  Completion forest of  $\mathcal{A}'_c$ ,  $\beta$  output mapping from SOLVE-MQ(...)  
**Output:**  $S$  set of pairs  $(a, b)$  s.t.  $(\mathcal{A}, \mathcal{T}, \mathcal{R}) \models R(a, b)$

- (1)  $DefnInfTriple \leftarrow RelInfTriple$  computed after evaluation of  $DefnRuleSet$  using  $\mathcal{T}, \mathcal{R}, \beta, \mathcal{A}'_c$  (for  $\mathcal{A}'$ ),  $\mathbf{f}_c$  (for  $\mathbf{f}$ )
- (2)  $S \leftarrow DefnInfTriple$
- (3)  $PotentialInfTriple \leftarrow RelInfTriple$  computed after evaluation of  $PotentialRuleSet$  using  $\mathcal{R}, \beta, \mathcal{A}'_c$  (for  $\mathcal{A}'$ ),  $\mathbf{f}_c$  (for  $\mathbf{f}$ ) and  $F'_c$  (for  $F'$ ) (Note: *Init* rule here initializes  $InfTriple$  as a union of role assertions in  $\mathcal{A}$  and  $DefnInfTriple$ )
- (4)  $PotentialInfTriple \leftarrow PotentialInfTriple - DefnInfTriple$  (remaining potential Abox role assertion solutions)
- (5) Test and Solve  $PotentialInfTriple$  as described in Section 6.3 to get solution pairs  $S'$
- (6)  $S \leftarrow S \cup S'$
- (7) **return**  $S$

## 7 Putting it all together

Finally, our complete conjunctive query answering algorithm, SOLVE-CQ, combines SOLVE-MQ and SOLVE-RQ.

SOLVE-CQ( $Q, \mathcal{A}, \mathcal{T}, \mathcal{R}$ )  
**Input:**  $Q$  Conjunctive query,  $\mathcal{A}$  Abox,  $\mathcal{T}$  Tbox,  $\mathcal{R}$  Rbox  
**Output:**  $S$  set of tuple solutions to  $Q$

- (1)  $[\mathcal{A}'_c, \mathbf{f}_c, F'_c, \beta] \leftarrow \text{SOLVE-MQ}(Q, \mathcal{A}, \mathcal{T}, \mathcal{R})$
- (2) **foreach**  $R(x_j, x_k)$  in  $Q$
- (3)  $S_{R_{j,k}} \leftarrow \text{SOLVE-RQ}(R(x_j, x_k), \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{A}'_c, \mathbf{f}_c, F'_c, \beta)$
- (4)  $S \leftarrow \text{join all relationship solutions in } S_{R_{j,k}} \text{ for all } R(x_j, x_k) \text{ in } Q$
- (5) **return**  $S$

## 8 Computational Experience

### 8.1 Correctness and Scalability tests

We evaluated our approach on the UOBM benchmark [8], which was modified to *SHIN* expressivity. We used 14 of the 15 queries defined in the benchmark (query Q2, which is a pure membership query, was not included in our evaluation. See appendix A for the 14 queries used here). The results are reported for 1, 5, 10, 30, 100 and 150 universities. We compared our results against KAON2

[9]. (Pellet v1.4 [10] did not scale to even one university). For KAON2, we set all maximum cardinality restrictions to one because of KAON2 limitations. Our experiments were conducted on a 2-way 2.4GHz AMD Dual Core Opteron system with 16GB of memory running Linux, and a maximum heap size of 2G. The Abox was stored in a IBM DB2 V9.1 for SHER and MySQL V5.0 for KAON2.

Dataset	type assertions	role assertions
1	25K	214K
5	120K	928K
10	224K	1,816K
30	709K	6.5M
100	7.8M	22.4M
150	11.7M	33.5M

(a) Dataset Statistics

Reasoner	Dataset	Avg. Time	St.Dev	Min-Max
KAON2	1	18	5	15-29
KAON2	5	166	102	111-487
KAON2	10	667	508	447-2319
SHER	1	12	2	8-15
SHER	5	25	6	16-35
SHER	10	46	14	27-71
SHER	30	150	50	88-228
SHER	100	531	322	278-1496
SHER	150	1066	706	465-3283

(b) Runtimes in sec

**Table 1.** Evaluation data

The size of the datasets are given in Table 1 (a). Table 1 (b) summarizes the times taken (in seconds) by KAON2 and SHER solely for query answering, i.e., in both cases, the times do not include the knowledge base pre-processing and setup costs. Figure 6, 7 and 8 compare SHER and KAON2 evaluation performance for dataset 1, 5 and 10. KAON2 ran out of memory on UOBM-30. In 13 out of 14 queries SHER and KAON2 had 100% agreement. The difference on query Q15 was due to differences in the constraints used. As can be seen, the average runtimes for SHER are significantly lower, usually by an order of magnitude, than those for KAON2.

As shown on Figure 9, on all queries, except query 9, SHER scales almost linearly from UOBM-1 to UOBM-150. Query 9, which has 3 role atoms, is an example of a query where we can improve our performance by using a cost model based approach to control the order of evaluation of query atoms as explained in [5].

## 8.2 Handling Non-deterministic Mergers

In experiments described in the previous subsection, UOBM queries did not exploit non-deterministic mergers between individuals in the Abox to produce new inferred results. Therefore, we modified the UOBM dataset to generate new relationships from non-deterministic mergers between named individuals, and considered a new query whose solutions required this.

We added disjoint relations between the four UOBM concepts **FineArts**, **Science**, **HumanitiesAndSocial**, **Engineering** representing course subjects, and a set of Abox assertions each resembling the pattern shown in Figure 10. The

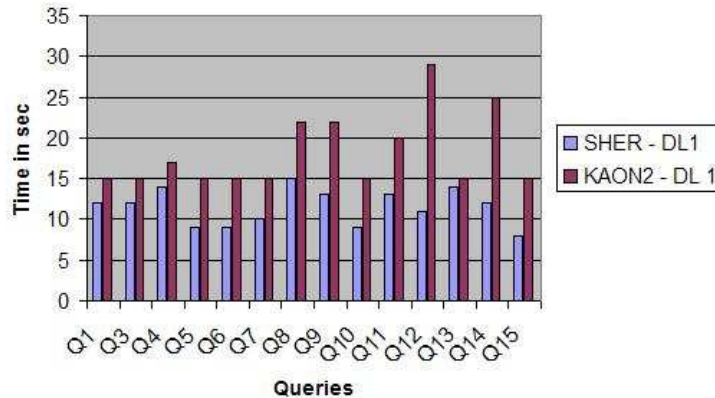


Fig. 6. SHER and KAON2 Comparison on UOBM-1

newly added individual  $LS_1$  had type `LeisureStudent`, which is defined as ( $\leq 3$ . `takesCourse`) in the UOBM Tbox.  $LS_1$  was assigned four `takesCourse` relations to individuals  $C_1..C_4$  respectively. In general, we randomly added any one of the four course subjects mentioned above as a type to  $C_i$ ,  $1 \leq i \leq 3$  ( $C_4$  is always assigned the type `Course`). In the case shown,  $C_1, C_2, C_3$  are mutually disjoint concepts and hence the `maxCardinality` restriction in the type of  $LS_1$  causes a non-deterministic merger between  $C_4$  and any one  $C_j$  ( $1 \leq j \leq 3$ ), which in turn causes  $C_4$  to acquire a new `isTaughtBy` relation to the `Lecturer` individual  $L_1$ . To exploit this behavior, we considered the query:  $Q_{ND}: (x, y, z) \leftarrow LeisureStudent(x) \wedge takesCourse(x, y) \wedge Course(y) \wedge isTaughtBy(y, z) \wedge Lecturer(z)$ . In the example shown, there are 4 tuple solutions to  $Q_{ND}$ , three of which are explicit ( $LS_1, C_1/C_2/C_3, L_1$ ), and one is inferred ( $LS_1, C_4, L_1$ ).

We modified UOBM-1, UOBM-5 and UOBM-10 by adding 100, 200 and 300 instances of `LeisureStudent` respectively. These numbers and datasets were chosen since as the pattern in Figure 10 shows, generation of new relationships due to non-deterministic mergers is non-trivial and seldom seen in large quantities in practice. We then evaluated  $Q_{ND}$  on the modified datasets. KAON2 is unable to handle this query since it cannot deal with non-deterministic mergers. Results of this query evaluation using SHER are shown in Figure 11. In the table, the column  $E$  (resp.  $I$ ) stands for the number of explicit (resp. inferred) solutions for the query introduced by our script,  $P_A$ , computed in step (4) of SOLVE-RQ, is the number of potential relationship pairs in the Abox that need to be tested,  $P_{A'}$  is the number of summary pairs corresponding to the Abox pairs counted in  $P_A$ , and  $S_{A'}$  is the number of summary solution tuples found using the procedure described in Section 6.3, which are eventually split down to the individual level.

As the results show, the algorithm demonstrates a graceful degradation for this query. For example, in UOBM-10, there are  $786+152 = 938$  entailed `isTaughtBy`

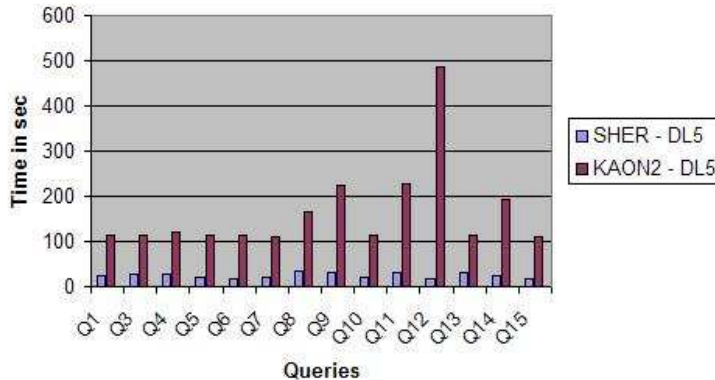


Fig. 7. SHER and KAON2 Comparison on UOBM-5

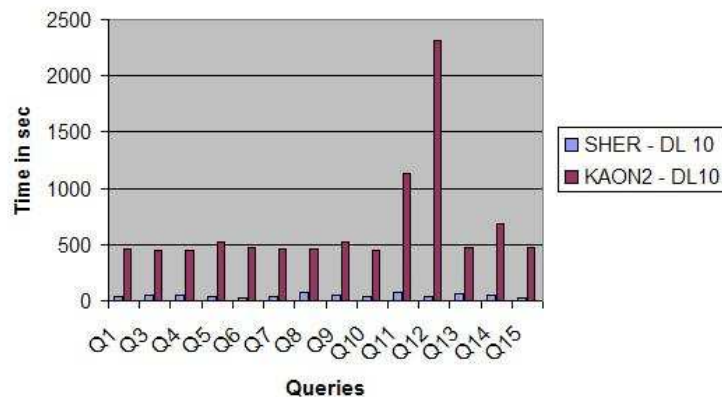
relationships (152 due to non-deterministic mergers<sup>6</sup>), however our algorithm finds, in step(4) of SOLVE-RQ, that only 319 need to be tested. Moreover, they are first tested through their corresponding summary pairs as explained in Section 6.3. As result, only 15 out of 100 summary pairs are found to be solutions<sup>7</sup>, and only one end of these 15 pairs are split down to the individual level. We feel that the times shown are acceptable for realistic use-cases.

## 9 Related Work & Conclusions

Scalable reasoning algorithms exist for Aboxes in secondary storage, but they either assume role-free Aboxes [11], or relatively inexpressive-DLs [6]. For the more expressive OWL-DL, state-of-the-art tableau reasoners such as Pellet and RACER have recently incorporated optimizations to support conjunctive query answering over large Aboxes ([5], [12]). Inspired by relational-database join optimizations, the systems use various heuristics to estimate a cost-model for evaluating the various query atoms in the conjunctive query, to determine an efficient join order. Additionally, the completion forest of the Abox (aka pseudo-model) is used to identify obvious solutions and non-solutions to a query. Finally, Tbox and Rbox information is used to rewrite conjunctive queries into a simpler form. However, a fundamental limitation is that they work with the complete Abox, and the complexity of the tableau reasoning algorithm makes it infeasible to build a completion forest for a large and expressive Abox, which affects both solution pruning and testing. For this reason, the current implementations of these systems scale to thousands but not millions of Abox assertions.

<sup>6</sup> Only `isTaughtBy` relations can be inferred due to non-deterministic mergers.

<sup>7</sup> Not all potential relationships are solutions since the script may not necessarily add disjoint subject types to individuals  $C_1, C_2, C_3$ .



**Fig. 8.** SHER and KAON2 Comparison on UOBM-10

On the other hand, KAON2, which we included in our evaluation, is a non-tableau based approach that relies on translating Description Logic to disjunctive datalog [13] and is able to scale to an Abox with a million assertions. However, KAON2 has problems dealing with max-cardinality restrictions (for cardinality greater than 1) and even excluding such restrictions, is unable to scale to an Abox with 7 million assertions.

Our technique appears to scale almost linearly for conjunctive queries of large, expressive Aboxes composed of 30-45 million Abox assertions. As future work, we plan to integrate a cost-model to determine an efficient join order for the query atoms.

## References

1. Dolby, J., A.Fokoue, Kalyanpur, A., A.Kershenbaum, L.Ma, E.Schonberg, K.Srinivas: Scalable semantic retrieval through summarization and refinement. Proc. of the 22nd Conf. on Artificial Intelligence (AAAI 2007) (2007)
2. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of owl dl entailments. In: Proc. of Int. Semantic Web Conf. (ISWC-2007). (2007)
3. Horrocks, I., Sattler, U., Tobies, S.: Reasoning with individuals for the description logic SHIQ\*. Proc. of 17th Int.Conf. on Automated Deduction (2000) 482-496
4. A.Fokoue, A.Kershenbaum, L.Ma, E.Schonberg, K.Srinivas: The summary abox: Cutting ontologies down to size. Proc. of the Int. Semantic Web Conf. (ISWC 2006) (2006) 136-145
5. Sirin, E., Parsia, B.: Optimizations for answering conjunctive abox queries: First results. In: Proc. of the Description Logics Workshop. (DL-06). (2006)
6. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: DL-lite: Tractable description logics for ontologies. Proc. of AAAI (2005)
7. Dolby, J., Fokoue, A., Kalyanpur, A., Ma, L., Schonberg, E., Srinivas, K., Sun, X.: Efficient reasoning on large shin aboxes in rela-

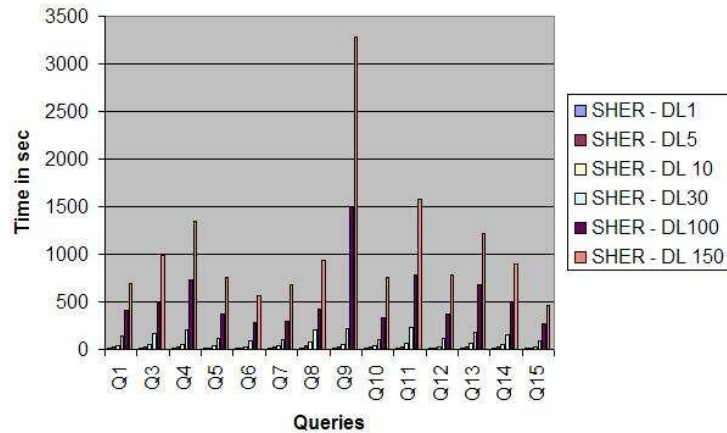


Fig. 9. SHER Scalability

- tional databases. In: [http://domino.research.ibm.com/comm/research\\_projects.nsf/pages/iaa.index.html/\\$FILE/ISWC08QEP.pdf](http://domino.research.ibm.com/comm/research_projects.nsf/pages/iaa.index.html/$FILE/ISWC08QEP.pdf). (2008)
8. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y.: Towards a complete owl ontology benchmark. In: Proc. of the third European Semantic Web Conf.(ESWC 2006). (2006) 124–139
  9. Hustadt, U., Motik, B., Sattler, U.: Reducing shiq - description logic to disjunctive datalog programs. Proc. of the 9th Int. Conf. on Knowledge Representation and Reasoning (KR 2004) (2004)
  10. Sirin, E., Parsia, B.: Pellet: An owl dl reasoner. In: Description Logics. (2004)
  11. Bechhofer, S., Horrocks, I., Turi, D.: The owl instance store: System description. Proc. of 20th Int.Conf. on Automated Deduction (2005) 177–181
  12. Moller, R., Haarslev, V., Wessel, M.: On the scalability of description logic instance retrieval. In: Proc. of the Description Logics Workshop. (DL-06). (2006)
  13. U.Hustadt, Motik, B., Sattler, U.: Reducing shiq description logic to disjunctive datalog programs. (Proc. of 9th Intl. Conf. on Knowledge Representation and Reasoning (KR2004)) 152–162

## A UOBM Queries

UOBM queries used in the experimental evaluation are listed below.

```
#Q1
SELECT DISTINCT ?X
WHERE {
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#UndergraduateStudent> .
?X <http://semantics.crl.ibm.com/univ-bench-dl.owl#takesCourse>
  <http://www.Department0.University0.edu/Course0>};
```

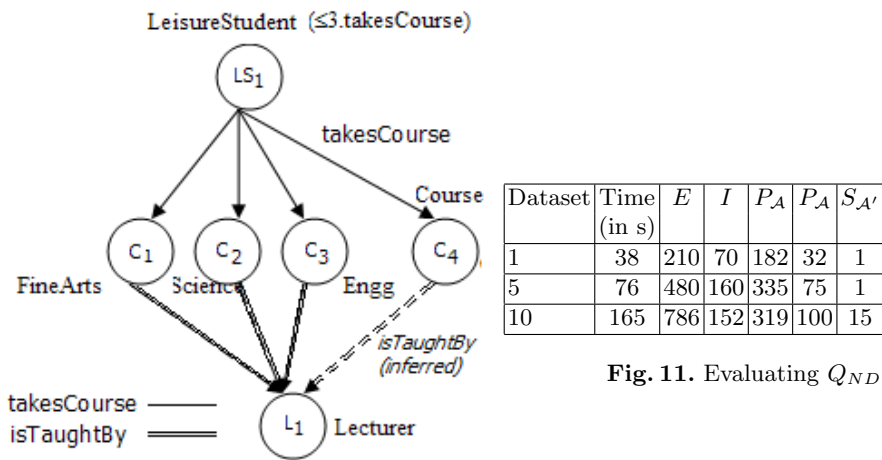


Fig. 11. Evaluating  $Q_{ND}$

Fig. 10. Abox pattern creating new isTaughtBy relations from non-deterministic mergers

```
#Q3
SELECT DISTINCT ?X
WHERE {
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://semantics.crl.ibm.com/univ-bench-dl.owl#Student> .
?X <http://semantics.crl.ibm.com/univ-bench-dl.owl#isMemberOf>
<http://www.Department0.University0.edu> };
```

```
#Q4
SELECT DISTINCT ?X
WHERE {
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://semantics.crl.ibm.com/univ-bench-dl.owl#Publication> .
?X <http://semantics.crl.ibm.com/univ-bench-dl.owl#publicationAuthor> ?Y .
?Y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://semantics.crl.ibm.com/univ-bench-dl.owl#Faculty> .
?Y <http://semantics.crl.ibm.com/univ-bench-dl.owl#isMemberOf>
<http://www.Department0.University0.edu> };
```

```
#Q5
SELECT DISTINCT ?X
WHERE {
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://semantics.crl.ibm.com/univ-bench-dl.owl#ResearchGroup> .
```

```
?X <http://semantics.crl.ibm.com/univ-bench-dl.owl#subOrganizationOf>
  <http://www.University0.edu> };
```

#Q6

```
SELECT DISTINCT ?X
WHERE {
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#Person> .
<http://www.University0.edu>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#hasAlumnus> ?X };
```

#Q7

```
SELECT DISTINCT ?X
WHERE {
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#Person> .
?X <http://semantics.crl.ibm.com/univ-bench-dl.owl#hasSameHomeTownWith>
  <http://www.Department0.University0.edu/FullProfessor0>;
```

#Q8

```
SELECT ?X
WHERE {
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#SportsLover> .
<http://www.Department0.University0.edu>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#hasMember> ?X .};
```

#Q9

```
SELECT DISTINCT ?X
WHERE {
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#GraduateCourse> .
?X <http://semantics.crl.ibm.com/univ-bench-dl.owl#isTaughtBy> ?Y .
?Y <http://semantics.crl.ibm.com/univ-bench-dl.owl#isMemberOf> ?Z .
?Z <http://semantics.crl.ibm.com/univ-bench-dl.owl#subOrganizationOf>
  <http://www.University0.edu> };
```

#Q10

```
SELECT DISTINCT ?X
WHERE {
?X <http://semantics.crl.ibm.com/univ-bench-dl.owl#isFriendOf>
  <http://www.Department0.University0.edu/FullProfessor0>;
```

#Q11

```
SELECT DISTINCT ?X
```

```
WHERE {
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#Person> .
?X <http://semantics.crl.ibm.com/univ-bench-dl.owl#like> ?Y .
?Z <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#Chair> .
?Z <http://semantics.crl.ibm.com/univ-bench-dl.owl#isHeadOf>
  <http://www.Department0.University0.edu> .
?Z <http://semantics.crl.ibm.com/univ-bench-dl.owl#like> ?Y};
```

#Q12

```
SELECT DISTINCT ?X
WHERE {
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#Student> .
?X <http://semantics.crl.ibm.com/univ-bench-dl.owl#takesCourse> ?Y .
?Y <http://semantics.crl.ibm.com/univ-bench-dl.owl#isTaughtBy>
  <http://www.Department0.University0.edu/FullProfessor0> };
```

#Q13

```
SELECT DISTINCT ?X
WHERE {
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#PeopleWithHobby> .
?X <http://semantics.crl.ibm.com/univ-bench-dl.owl#isMemberOf>
  <http://www.Department0.University0.edu>;
```

#Q14

```
SELECT ?X
WHERE {
?X <http://semantics.crl.ibm.com/univ-bench-dl.owl#isMemberOf> ?Y .
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#Woman> .
?Y <http://semantics.crl.ibm.com/univ-bench-dl.owl#subOrganizationOf>
  <http://www.University0.edu> .
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#Student> .};
```

#Q15

```
SELECT DISTINCT ?X
WHERE {
?X <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://semantics.crl.ibm.com/univ-bench-dl.owl#PeopleWithManyHobbies> .
?X <http://semantics.crl.ibm.com/univ-bench-dl.owl#isMemberOf>
  <http://www.Department0.University0.edu> };
```