Scalable Grounded Conjunctive Query Evaluation over Large and Expressive Knowledge Bases

Julian Dolby[1], Achille Fokoue[1], Aditya Kalyanpur[1], Li Ma[2], Edith Schonberg[1], Kavitha Srinivas[1], and Xingzhi Sun[2]

[1] IBM Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA
{dolby,achille,adityakal,ediths,ksrinivs}@us.ibm.com
[2] IBM China Research Lab, Beijing 100094, China
{malli,sunxingz}@cn.ibm.com

Abstract. Grounded 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 grounded 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. In this paper, we present a tableau-based reasoning solution for answering grounded conjunctive queries over large and expressive Aboxes. Grounded conjunctive queries, which use distinguished variables only, are more realistic in practice and can also be answered more efficiently than the more general case. Our approach is sound and complete for DL $\mathcal{SHIN}$ (OWL-DL minus nominals and datatypes).

The naive tableau-based algorithm for grounded conjunctive query is to split the query into its component membership atoms and relationship atoms, solve each atom separately, and join the respective bindings at the end. For example, consider a conjunctive query $C(x) \wedge R(x, y) \wedge D(y)$ where x and y are distinguished variables, C and D are concepts, and R and S are roles. Naively, the membership atoms $C(x)$ and $D(x)$ and the relationship atom $R(x, y)$ are solved as three separate queries, and then the result bindings are joined. Without any optimization, solving each membership atom requires testing every individual in the Abox, and solving each relationship query requires testing every pair of individuals in the Abox. For the membership atom $C(x)$, for each individual a, the assertion $a : \neg C$ is added to the Abox, and the new Abox is tested for consistency. For the relationship atom $R(x, y)$, for each pair of individuals a and b, the assertions $a : \neg\exists R N_b$ and $b : N_b$ are added to the Abox, where N_b is a new concept. This abox is then tested for consistency. Optimizations reduce the number of tests that need to be performed. However, this approach remains fundamentally impractical for large Aboxes.

This paper builds on our previous technique for solving membership queries over $\mathcal{SHIN}$ KBs containing millions of assertions [1]. 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 (i.e., the same set of explicit types) into a single summary individual (of that same type). For a given membership query, its negation is added to the concept set of each individual a in $\mathcal{A}'$. If the summary is consistent, then all individuals mapped to a can be ruled out as solutions to the query. If 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 the individuals in the summary Abox after each refinement step, even individuals in precise justifications, still represent many individuals in the original Abox. The scalability of the approach comes from the fact that it makes decisions on groups of individuals as a whole in the summary.

We extend our summarization-based technique to solve grounded conjunctive queries. Like the naive tableau-based conjunctive algorithm, we split a conjunctive query into its atomic parts, solve each atom separately, and join the respective bindings at the end. To find bindings for membership atoms, we apply our membership algorithm on the summary Abox (Section 3). The relationship atoms provide additional optimization opportunities: potential membership query bindings that do not satisfy relationship constraints are filtered out as candidates. We use the completion forest of the summary Abox to filter candidates, and reduce the search space significantly. We prove that this optimization is correct for summary Aboxes in Section 3.1.

Solving relationship atoms efficiently using the summary Abox is not as straightforward. For a relationship atom $R(x, y)$ and $\mathcal{A}'$ individuals a and b, we cannot simply add $a : \neg\exists R N_b$ to $\mathcal{A}'$ and apply our summary Abox membership algorithm. This is because both a and b in $\mathcal{A}'$ each represent many individuals in $\mathcal{A}$. If a and b satisfy this relationship atom, we need to find all pairs a_i, b_i in $\mathcal{A}$ that map to a and b respectively, such that $R(a_i, b_i)$ holds in $\mathcal{A}$. This requires testing some individual pairs in the original Abox for completeness, which defeats the advantage of summarization[1]. Instead, we apply datalog reasoning over the original Abox to conservatively estimate candidates, and apply a modified summary graph algorithm to determine which of these candidates are real solutions. We present details in Section 4.

[1] For details, see [2].

Our contributions in this paper are as follows: (a) we present a technique to perform scalable grounded 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

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 $(x_1, ..., x_n) \leftarrow q_1 \wedge ... \wedge q_m$ 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 or named individuals in $\mathcal{K}$, C is a concept expression and R is a role. $Var(Q)$ refers to the set of variables 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 $\pi(x)$.

$(a_1, ..., a_n)$ is a solution in the KB $\mathcal{K}$ of the conjunctive query Q of the form $(x_1, ..., x_n) \leftarrow q_1 \wedge ... \wedge q_m$ 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 $\pi.q_j$).

2.2 Summarization and Refinement

In our earlier work, we presented an algorithm based on summarization and refinement to scale consistency checking 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 [3]. 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 $\mathcal{SHIN}$[2] Abox $\mathcal{A}'$ if there is a mapping function $\mathbf{f}$ that satisfies the following constraints:

(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 $a \dot{\neq} b \in \mathcal{A}$ then $\mathbf{f}(a) \dot{\neq} \mathbf{f}(b) \in \mathcal{A}'$

[2] We assume without loss of generality that $\mathcal{A}$ does not contain an assertion of the form $a \doteq b$

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.

2.3 Tableau Completion Forest

As described in [4], the tableau algorithm operates on a completion forest $F = (G, \mathcal{L}, \dot{\neq}, \doteq)$ 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 $< x, y >$ in G to a set of roles, $\mathcal{L}(< x, y >)$, in $\mathcal{R}$; $\doteq$ is an equivalence relation corresponding to the equality between nodes of G; and $\dot{\neq}$ is the binary relation *distinct from* on nodes of G. At the beginning of the execution of the tableaux 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 $\mathcal{A}$. $< x, y >$ is an edge in G with $R \in \mathcal{L}(< x, y >)$ iff $R(x, y) \in \mathcal{A}$. For x and y in G, $x \dot{\neq} y$ iff $x \dot{\neq} y \in \mathcal{A}$. Initially, there are no x and y in G such that $x \doteq y$. The tableaux 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):

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

3 Solving the Membership Query Part

The algorithm for solving membership queries on the summary Abox consists of two phases: finding candidate individuals in the summary Abox, and then applying the membership query algorithm as described in [1] to all of the candidates. The technique for finding candidates is described in Section 3.1. The complete algorithm is described in Section 3.2.

3.1 Optimizing Conjunctive Querying with the Summary Completion Forest

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[3] 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 below.

By theorem 1, if F' denotes the clash-free completion forest resulting from the consistency check on the summary $\mathcal{A}'$ of $\mathcal{A}$, then there exists a complete and 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 $\mathcal{A}$ and a role S in $\mathcal{R}$,*

(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').

[3] By definition, y is a R-neighbor of x iff. $S(x, y) \in \mathcal{A}$ or $P(y, x) \in \mathcal{A}$ where S and P^- are subroles of R

– Second, we maintain, during the execution of the tableau algorithm on $\mathcal{A}$, a mapping σ 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 σ 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 σ 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 σ is extended to these new nodes as explained 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[4] 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 $x \dot{\neq} y$ holds in F, then $\sigma(x)[2] \dot{\neq} \sigma(y)[2]$ 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.

The details of the induction proof is given in [2].

3.2 Membership Query Algorithm

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.

[4] An execution in which non-deterministic choices are made based on choices made in F' as explained in the treatment of non-deterministic rules

```
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, $\mathbf{f}$ Abox 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 $\tau(x \mapsto i)$
```

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 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 τ 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 τ.

Having identified suitable test candidates, we now proceed to test them for their respective membership query atoms, using our summarization and refinement algorithm [1].While the previous work focused on testing a single membership query on the summary, it can be easily extended to test multiple membership queries on the summary at the same time. The main difference is that we now start by adding the negation of all the membership types to their respective summary individual candidates, before testing for inconsistency (for details of other optimizations to membership querying, see [2]).

SOLVE-MQ, sketched below, captures the essence of the evaluation of membership queries.

```
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 in-
              dividuals in $\tau(x_k)$ instances of concept $\bigcap_{C_p(x_k) \in Q} Cp$
(8)       else
(9)           $\beta(x_k) \leftarrow \tau(x_k)$
(10)  return $(\mathcal{A}'_c$ , $\mathbf{f}_c$, $F'_c$, $\beta)$
```

4 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 4.1: 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 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 4.2: 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 4.3: Finally, we test and solve the remaining potential relationship solutions in the summary Abox.

4.1 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

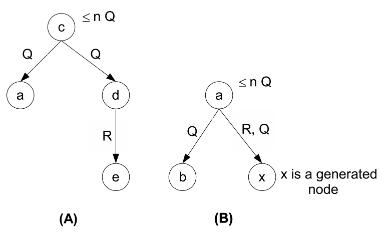

Fig. 1. Acquisition of named individual R-neighbors

(Init)	*InfTriple(X, R, Y)*	:- $R(X, Y) \in \mathcal{A}$
(SameSym)	*same(X, Y)*	:- *same(Y, X)*
(SameTrans)	*same(X, Y)*	:- *same(X,Z) and same(Z,Y)*
(NamedMerge)	*same(X, Y)*	:- $\mathbf{f}(Z) = A$ *and* $\leq nR \in \mathcal{L}'(\alpha(A))$ *and* $X \neq Y$ *and InfTriple(Z, R, X) and InfTriple(Z, R, Y)*
(SameRel1)	*InfTriple(X, R, Y)*	:- *same(X,Z) and InfTriple(Z, R, Y)*
(SameRel2)	*InfTriple(X, R, Y)*	:- *same(Y,Z) and InfTriple(X, R, Z)*
(UnnamedMerge)	*InfTriple(X, R, Y)*	:- $\mathbf{f}(X) = A$ *and* $\mathbf{f}(Y) = B$ *and* $\leq nT \in \mathcal{L}'(\alpha(A))$ *and* $(\exists S.C \in \mathcal{L}'(\alpha(A))$ *or* $\geq mS \in \mathcal{L}'(\alpha(A)))$ *and* $\alpha(B)$ *is a R-neigbhor of* $\alpha(A)$ *in* F' *and* $\{R, T\} \subseteq \widehat{\phi_A}$ *and InfTriple(X, T, Y)*
(SubRole)	*InfTriple(X, R, Y)*	:- $S \sqsubseteq^* R$ *and InfTriple(X, S, Y) and* $S \neq R$
(InvRole)	*InfTriple(X, R, Y)*	:- $S^- = R$ *and InfTriple(Y, S, X)*
(Relevance)	*RelInfTriple(X, R, Y)*	:- *InfTriple(X, R, Y) and* $\mathbf{f}(X) = A$ *and* $\mathbf{f}(Y) = B$ *and* $R(x_1, x_2) \in Q$ *and* $A \in \beta(x_1)$ *and* $B \in \beta(x_2)$

Fig. 2. PotentialRuleSet: Rules to compute potential new named individual neighbors that are relevant to conjunctive query Q. Main output: $RelInfTriple$.

present before the beginning of rule execution). Then, we devise a set of simple rules (see Figure 2) to conservatively estimate potential R-neighbors. These rules are simple enough to be efficiently evaluated using a datalog engine. Figure 1 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 $NamedMerge$, $SameRel1$, and $SameRel2$ in Figure 2) 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 1 (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-neigbhor of a, and
  - x must be a Q-neigbhor 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 ϕ_a of roles P such that there is a generated node x P-neighbor of a in F, since R has to be in ϕ_a. A direct consequence of property (B') in the proof of Theorem 1 is that the following set is such an upper bound: $\{P|$ there is a P-neigbhor of $\alpha(\mathbf{f}(a))$ in $F'\}$.

Let $\widehat{\phi_{\mathbf{f}(a)}}$ be an upper bound of the set ϕ_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-neigbhor 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 2 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 neigbhors (computed by rules in Figure 2). It is important to note that new relations found after the application of the transitive closure cannot cause merger rules to trigger because, in $\mathcal{SHIN}$, 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 2 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 β in the output of algorithm SOLVE-MQ.

4.2 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 2 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 $\leq 1.R(x)$, 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 3, which replace the rules $NamedMerge$, $UnnamedMerge$ in the $PotentialRuleSet$ (Figure 2)

(SummaryKB Defn)	$\mathcal{K}$	$= (\mathcal{A}, \mathcal{T}, \mathcal{R})$
(DefnNamedMerge)	*same(X, Y)*	:- $\mathcal{K} \models \leq 1R(Z)$ *and* $X \neq Y$
		and InfTriple(Z, R, X)
		and InfTriple(Z, R, Y)
(DefnUnnamedMerge)	*InfTriple(X, R, Y)*	:- $\mathcal{K} \models \leq 1T(X)$
		and ($\mathcal{K} \models \exists S.C(X)$ *or* $\mathcal{K} \models \geq mS(X)$)
		and $S \sqsubseteq^* R$ *and* $S \sqsubseteq^* T$
		and InfTriple(X, T, Y)

Fig. 3. DefnRuleSet: Obtained by replacing $NamedMerge$ and $UnnamedMerge$ in the PotentialRuleSet with the rules shown

to produce the rule set $DefnRuleSet$ that computes definite Abox relationship solutions.

4.3 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 $\{R(u_1, v_1), ...R(u_n, v_n)\}$, 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)[5].

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 SOLVE-RQ that finds all solutions to a relationship query.

[5] From a precise summary justification for $R(b_i, b_j)$, we can issue an SQL query based on the justification pattern to get some relationship pair solutions in the Abox, but this would not be complete. For details, see [2]

```
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 $A'_c$, $\beta$ 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 evalua-
       tion 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 (re-
       maining potential Abox role assertion solutions)
(5)    Test and Solve PotentialInfTriple as described in Section
       4.3 to get solution pairs $S'$
(6)    $S \leftarrow S \cup S'$
(7)    return $S$
```

5 Computational Experience

5.1 Correctness and Scalability Tests

We evaluated our approach on the UOBM benchmark [6], which was modified to $\mathcal{SHIN}$ 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). The results are reported for 1, 5, 10, 30, 100 and 150 universities. We compared our results against KAON2 [7]. (Pellet [8] 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.

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. 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. [2] presents more detailed data on the evaluation performance for each query on each KB. 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].

Table 1. Evaluation data

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	Range
KAON2	1	18	5	14
KAON2	5	166	102	376
KAON2	10	667	508	1872
SHER	1	12	2	7
SHER	5	25	6	19
SHER	10	46	14	44
SHER	30	150	50	140
SHER	100	531	322	1222
SHER	150	1066	706	2818

(b) Runtimes in sec

5.2 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 4. The 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 4 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 5. In the table, the column E (resp. I) stands for the number of explicit (resp. inferred) solutions for the query introduced by our script, $P_{\mathcal{A}}$, computed in step (4) of SOLVE-RQ, is the number of potential relationship pairs in the Abox that need to be

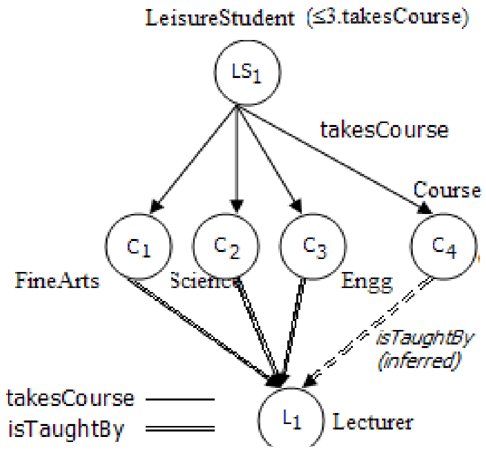

Fig. 4. Abox pattern creating new `isTaughtBy` relations from non-deterministic mergers

Dataset	Time (in s)	E	I	$P_{\mathcal{A}}$	$P_{\mathcal{A}}$	$S_{\mathcal{A}'}$
1	38	210	70	182	32	1
5	76	480	160	335	75	1
10	165	786	152	319	100	15

Fig. 5. Evaluating Q_{ND}

tested, $P_{\mathcal{A}'}$ is the number of summary pairs corresponding to the Abox pairs counted in $P_{\mathcal{A}}$, and $S_{\mathcal{A}'}$ is the number of summary solution tuples found using the procedure described in Section 4.3, 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 a total of 786+152 = 938 entailed `isTaughtBy` relationships (152 due to non-deterministic mergers[6]), 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 4.3. As result, only 15 out of 100 summary pairs are found to be solutions[7], 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.

6 Related Work and Conclusions

We have focused on answering grounded conjunctive queries instead of general conjunctive queries because, to our knowledge, there is currently no practical algorithm for answering general conjunctive queries with respect to SHIN ontologies [9]. It is for this reason that OWL-DL reasoner implementations which support conjunctive query answering, such as Pellet [8], RACER [10] and KAON2 [7], do so with the grounded conjunctive query semantics.At the same time, even after using the grounded conjunctive query semantics, tableau-based

[6] Only `isTaughtBy` relations can be inferred due to non-deterministic mergers.

[7] Not all potential relationships are solutions since the script may not necessarily add disjoint subject types to individuals C_1, C_2, C_3.

reasoners such as Pellet and RACER do not scale to several millions of assertions as SHER does. The 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.

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 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.

In our experiments, our technique appears to scale almost linearly for conjunctive queries of large, expressive Aboxes composed of 30-45 million Abox assertions, and conceptually, nothing in our approach prevents it from scaling to much larger datasets. As future work, we plan to integrate a cost-model to determine an efficient join order for the query atoms.

References

1. Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Ma, L., Schonberg, E., Srinivas, K.: Scalable semantic retrieval through summarization and refinement. In: Proc. of the 22nd Conf. on Artificial Intelligence, AAAI 2007 (2007)
2. Dolby, J., Fokoue, A., Kalyanpur, A., Ma, L., Schonberg, E., Srinivas, K., Sun, X.: Scalable conjunctive query evaluation: Technical report (2008), `http://domino.research.ibm.com/comm/research_projects.nsf/pages/iaa.index.html/$FILE/techReportCQ.pdf`
3. Fokoue, A., Kershenbaum, A., Ma, L., Schonberg, E., Srinivas, K.: The summary abox: Cutting ontologies down to size. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 136–145. Springer, Heidelberg (2006)
4. Horrocks, I., Sattler, U., Tobies, S.: Reasoning with individuals for the description logic SHIQ*. In: Proc. of 17th Int.Conf. on Automated Deduction, pp. 482–496 (2000)
5. Sirin, E., Parsia, B.: Optimizations for answering conjunctive abox queries: First results. In: Proc. of the Description Logics Workshop, DL 2006 (2006)
6. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y.: Towards a complete owl ontology benchmark. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 124–139. Springer, Heidelberg (2006)
7. Hustadt, U., Motik, B., Sattler, U.: Reducing shiq description logic to disjunctive datalog programs. In: Proc. of 9th Intl. Conf. on Knowledge Representation and Reasoning (KR 2004), pp. 152–162 (2004)
8. Sirin, E., Parsia, B.: Pellet: An OWL DL reasoner. In: Description Logics (2004)
9. Glimm, B., Horrocks, I., Lutz, C., Sattler, U.: Conjunctive query answering for the description logic shiq. In: IJCAI, pp. 399–404 (2007)
10. Haarslev, V., Moller, R.: Racer system description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 701–705. Springer, Heidelberg (2001)