# Secrecy-preserving Reasoning in $\mathcal{ELH}$ Knowledge Bases using MapReduce Algorithm

Gopalakrishnan Krishnasamy Sivaprakasam[1] and Giora Slutzki[2]

[1]*Department of Mathematics & Computer Science, Central State University, Wilberforce, Ohio, U.S.A.*

[2]*Department of Computer Science, Iowa State University, Ames, Iowa, U.S.A.*

Keywords: Secrecy Preserving Reasoning, Knowledge Bases, MapReduce Algorithm.

Abstract: In this paper, we have used the MapReduce algorithm to study the problem of secrecy-preserving reasoning in a very large $\mathcal{ELH}$ knowledge bases. A tableau algorithm for ABox reasoning is designed in a way that is suitable for MapReduce framework and contains a small set of reasoning rules. To implement the parallelization method, we have designed map and reduce functions for each of these ABox reasoning rules. The output of this computational procedure is a finite set $\mathcal{A}^*$ which contains assertional consequences of the given knowledge base. Given a finite secrecy set $\mathbb{S}$, we compute a set $\mathbb{E}$, called an envelope of $\mathbb{S}$, which provides logical protection to the secrecy set $\mathbb{S}$ against the reasoning of a querying agent. To compute $\mathbb{E}$, a tableau algorithm is designed by inverting ABox reasoning rules in a way that is suitable for MapReduce framework. Further, to implement the parallelization method, we have designed map and reduce functions for each of these inverted rules.

## 1 INTRODUCTION

In the last three decades, beginning with the internet era, there has been a dramatic increase in the volume of mobile devices and internet users whose daily web activities include online banking activities, social networking, web based travel services and other internet based business applications. These activities contribute to the generation of unprecedented amount of web data. In (Reinsel et al., 2017), the authors reported that estimated amount of web data would be around 17 zettabytes and they forecasted that the volume of web data would grow upto 160 zettabytes by 2025. Web data contains a massive amount of private information about users, administrators, service providers, governmental agencies and military establishments which must be maintained and protected. To deal with this emerging challenge it is imperative to design and develop robust Bigdata technology tools to secure the confidential information in ways that do not deter the main web objective of sharing the non-confidential information among users.

The logic based languages like Resource Description Framework (RDF) and Web Ontology Languages (OWL) have been developed exclusively for creating knowledge bases (ontologies) and designing efficient procedures to reasoning and answering queries on these web databases. RDF and OWL are prescribed languages for Semantic web technologies recommended by World Wide Web consortium (W3C). Recently, several software tools like Bigdata®, LargeTripleStores and Blazegraph™ have been developed specifically for Bigdata applications, and these tools support RDF and OWL formats. In addition, a number of research works have been reported on Big data applications in Semantic Web using RDF and OWL approaches, see (Konstantopoulos et al., 2016; Zhou et al., 2013; Hitzler and Janowicz, 2013).

Description Logic (DL) is a decidable fragment of First Order Logic (FOL). In Semantic web research, DL languages play a central role in building Knowledge Bases (KBs) for social, biological and medical applications, and reasoning and answering queries over KBs. Further, DL languages are considered to be underlying logics of OWLs (Krötzsch, 2012) which are recommended by the W3C as a knowledge representation and reasoning languages for the web. In recent years, there has been a widespread interest in Bigdata research. In particular, DL research community has also shown an increaseing interest in studying the problem of reasoning in DL KBs with Bigdata applications, see (Möller et al., 2013; Mutharaju et al., 2010; Bellomarini et al., 2018; Urbani et al., 2010).

Sivaprakasam, G. and Slutzki, G.
Secrecy-preserving Reasoning in ELH Knowledge Bases using MapReduce Algorithm.
DOI: 10.5220/0008986007080716
In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020) - Volume 2*, pages 708-716
ISBN: 978-989-758-395-7; ISSN: 2184-433X

Tao et al., in (Tao et al., 2015) have developed a conceptual framework to study Secrecy-Preserving reasoning and Query Answering (SPQA) in Description Logic (DL) Knowledge Bases (KBs) under Open World Assumptions (OWA). The approach uses the notion of an *envelope* to hide secret information against logical inference and it was first defined and used in (Tao et al., 2010). The idea behind the envelope concept is that no assertion in the envelope can be logically deduced from information outside the envelope. This approach is based on the assumption that the information contained in a KB is incomplete (by OWA). Specifically, in (Tao et al., 2010; Tao et al., 2015; Sivaprakasam and Slutzki, 2016; Sivaprakasam and Slutzki, 2017) the main idea was to utilize the secret information within the reasoning process, but then answering "Unknown" whenever the answer is truly unknown or in case the true answer could compromise confidentiality / secrecy.

In this paper, we study the SPQA problem on very large scale $\mathcal{ELH}$ KB, see (Krötzsch, 2012), using MapReduce approach. MapReduce is a distributed programming paradigm for processing very large data sets (Bigdata) in a distributed way over cluster of machines (Dean and Ghemawat, 2008; Karloff et al., 2010). The central idea behind this procedure is to define *map* and *reduce* functions, and compute the reduce function in a parallel manner. It is a very popular computational model used in Bigdata applications, for more details see section 3.1. A first step in designing SPQA system is to compute a restricted assertional inference closure $\mathcal{A}^*$ using MapReduce procedure. For this purpose, we make use of non-modalized version of a tableau algorithm whose proof of correctness is reported in (Sivaprakasam and Slutzki, 2017). For each Abox expansion rule in this tableau algorithm, depending on the keys, we define map and reduce functions. Each map-reduce computational cycle results in a parallel application of one of the expansion rules.

After computing $\mathcal{A}^*$, the next step is to compute an envelope $\mathbb{E}$ to protect the secret information given in the provided secrecy set $\mathbb{S}$. This envelope is computed by a another tableau algorithm based on the idea of inverting the ABox expansion rules given in the first tableau algorithm. For each of these inverted expansion rule, the customized map and reduce functions will be defined. Once such envelope is computed, the answers to the queries are censored whenever the queries depend upon the set $\mathbb{E}$. Since the set $\mathcal{A}^* \setminus \mathbb{E}$ does not contain all the statements entailed by $\Sigma$, we use a recursive query answering procedure similar to a procedure reported in (Sivaprakasam and Slutzki, 2016; Sivaprakasam and Slutzki, 2017) to answer more complicated queries.

## 2 SYNTAX AND SEMANTICS

A vocabulary of $\mathcal{ELH}$ is a triple $< N_O, N_C, N_R >$ of countably infinite, pairwise disjoint sets. The elements of $N_O$ are called *object* (or *individual) names*, the elements of $N_C$ are called *concept names* and the elements of $N_R$ are called *role names*. The set of $\mathcal{ELH}$ *concepts* is denoted by $\mathcal{C}$ and is defined by the following rules

$$C ::= A \mid \top \mid C \sqcap D \mid \exists r.C$$

where $A \in N_C$, $r \in N_R$, $\top$ denotes the "*top concept*", and $C, D \in \mathcal{C}$. *Assertions* are expressions of the form $C(a)$ or $r(a,b)$, *general concept inclusions (GCIs)* are expressions of the form $C \sqsubseteq D$ and *role inclusions* are expressions of the form $r \sqsubseteq s$ where $C, D \in \mathcal{C}$, $r, s \in N_R$ and $a, b \in N_O$. The semantics of $\mathcal{ELH}$ concepts is specified, as usual, by an *interpretation* $I = \langle \Delta, \cdot^I \rangle$ where $\Delta$ is the *domain* of the interpretation, and $\cdot^I$ is an *interpretation function* mapping each $a \in N_O$ to an element $a^I \in \Delta$, each $A \in N_C$ to a subset $A^I \subseteq \Delta$, and each $r \in N_R$ to a binary relation $r^I \subseteq \Delta \times \Delta$. The interpretation function $\cdot^I$ is extended inductively to all $\mathcal{ELH}$ concepts in the usual manner:

$$\top^I = \Delta; \ (C \sqcap D)^I = C^I \cap D^I;$$
$$(\exists r.C)^I = \{d \in \Delta \mid \exists e \in C^I : (d,e) \in r^I\}.$$

An *Abox* $\mathcal{A}$ is a finite, non-empty set of assertions. A *TBox* $\mathcal{T}$ is a finite set of GCIs and an *RBox* $\mathcal{R}$ is a finite set of role inclusions. An $\mathcal{ELH}$ KB is a triple $\Sigma = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where $\mathcal{A}$ is an ABox, $\mathcal{T}$ is a TBox and $\mathcal{R}$ is an RBox. Let $I = \langle \Delta, \cdot^I \rangle$ be an interpretation, $C, D \in \mathcal{C}$, $r, s \in N_R$ and $a, b \in N_O$. We say that $I$ *satisfies* $C(a)$, $r(a,b)$, $C \sqsubseteq D$ or $r \sqsubseteq s$, notation $I \models C(a)$, $I \models r(a,b)$, $I \models C \sqsubseteq D$ or $I \models r \sqsubseteq s$ if, respectively, $a^I \in C^I$, $(a^I, b^I) \in r^I$, $C^I \subseteq D^I$ or $r^I \subseteq s^I$. $I$ is a *model* of $\Sigma$, notation $I \models \Sigma$, if $I$ satisfies all the assertions in $\mathcal{A}$, all the GCIs in $\mathcal{T}$ and all the role inclusions in $\mathcal{R}$. Let $\alpha$ be an assertion, a GCI or a role inclusion. We say that $\Sigma$ *entails* $\alpha$, notation $\Sigma \models \alpha$, if all models of $\Sigma$ satisfy $\alpha$.

## 3 COMPUTATION OF A MODEL FOR $\mathcal{ELH}$ KB $\Sigma$ AND $\mathcal{A}^*$

Let $\Sigma = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ be an $\mathcal{ELH}$ KB. Denote by $N_\Sigma$ the set of all concept names and role names occurring in $\Sigma$ and let $\mathbb{S}$ be a finite set[1] of concepts over the

[1] A technicality; $\mathbb{S}$ will be used in section 4 in the context of secrecy-preserving reasoning.

symbol set $N_\Sigma$. Let $\mathcal{C}_{\Sigma,\mathbb{S}}$ be the set of all subconcepts of concepts that occur in either $\mathbb{S}$ or $\Sigma$ and define the set of assertions

$$\begin{aligned}\mathcal{A}^* = \{&C(a) \mid C \in \mathcal{C}_{\Sigma,\mathbb{S}} \text{ and } \Sigma \models C(a)\} \cup \\ &\{r(a,b) \mid r \text{ occurs in } \Sigma \text{ and } \Sigma \models r(a,b)\}.\end{aligned}$$

We use $\mathcal{O}_\Sigma$ to denote the set of individual names that occur in $\Sigma$, and define a fresh set of constants called as *witness set* $\mathcal{W} = \{w^r_C \mid r \in N_R \cap N_\Sigma \text{ and } C \in \mathcal{C}_{\Sigma,\mathbb{S}}\}$. This witness set plays an important role in designing a rule in ABox reasoning algorithm that splits an assertion of the form $\exists r.C(a)$ into two assertions $r(a, w^r_C)$ and $C(w^r_C)$. Further define $O^* = \mathcal{O}_\Sigma \cup \mathcal{W}$ and $AX = \{\top(a) \mid a \in O^*\}$.

$$\boxed{\begin{array}{rl}
\sqcap^+\text{-rule}: & \text{if } C(a),\ D(a) \in \mathcal{A}^*,\ C \sqcap D \in \mathcal{C}_{\Sigma,\mathbb{S}} \text{ and} \\
& C \sqcap D(a) \ \notin \mathcal{A}^*, \\
& \text{then } \mathcal{A}^* := \mathcal{A}^* \cup \{C \sqcap D(a)\}; \\
\sqcap^-\text{-rule}: & \text{if } C \sqcap D(a) \in \mathcal{A}^* \text{ and} \\
& C(a) \notin \mathcal{A}^* \text{ or } D(a) \ \notin \mathcal{A}^*, \\
& \text{then } \mathcal{A}^* := \mathcal{A}^* \cup \{C(a), D(a)\}; \\
\exists^+\text{-rule}: & \text{if } r(a,b),\ C(b) \in \mathcal{A}^*,\ \exists r.C \in \mathcal{C}_{\Sigma,\mathbb{S}} \text{ and} \\
& \exists r.C(a) \ \notin \mathcal{A}^*, \\
& \text{then } \mathcal{A}^* := \mathcal{A}^* \cup \{\exists r.C(a)\}; \\
\exists^-\text{-rule}: & \text{if } \exists r.C(a) \in \mathcal{A}^* \text{ and} \\
& \forall b \in O^*,\ \{r(a,b), C(b)\} \nsubseteq \mathcal{A}^*, \\
& \text{then } \mathcal{A}^* := \mathcal{A}^* \cup \{r(a, w^r_C), C(w^r_C)\}, \\
& \text{where } w^r_C \in \mathcal{W}; \\
\sqsubseteq\text{-rule}: & \text{if } C(a) \in \mathcal{A}^*,\ C \sqsubseteq D \in \mathcal{T} \text{ and} \\
& D(a) \ \notin \mathcal{A}^*, \\
& \text{then } \mathcal{A}^* := \mathcal{A}^* \cup \{D(a)\}; \\
H\text{-rule}: & \text{if } r(a,b) \in \mathcal{A}^*,\ r \sqsubseteq s \in \mathcal{R} \text{ and} \\
& s(a,b) \ \notin \mathcal{A}^*, \\
& \text{then } \mathcal{A}^* := \mathcal{A}^* \cup \{s(a,b)\}.
\end{array}}$$

Figure 1: ABox Tableau expansion rules.

Tableau algorithm given in Figure 1 matches exactly with non-modalized (without modal operator) version of the ABox tableau algorithm reported in (Sivaprakasam and Slutzki, 2017). In that paper, the authors discussed in detail the proof of correctness of modalized version of ABox tableau algorithm. Given $\Sigma$ and $C_{\Sigma,\mathbb{S}}$, this algorithm computes $\mathcal{A}^*$. $\mathcal{A}^*$ is initialized as $A \cup AX$ and is expanded by exhaustively applying expansion rules listed in Figure 1. The worst case running time of the algorithm given in Figure 1 is $O((\mid \Sigma \mid + \mid C_{\Sigma,\mathbb{S}} \mid)^3)$, for more details see Theorem 27 in (Krötzsch, 2012).

## 3.1 MapReduce Paradigm

In this paper, our goal is to cast ABox reasoning algorithm given in Figure 1 in MapReduce framework. First let us give a brief account of MapReduce procedure. MapReduce is a distributed computational model for processing data in parallel on clusters of machines. Typically, these machines are called as 'nodes', (Dean and Ghemawat, 2008; Karloff et al., 2010). The data set is partitioned into several subsets, and each subset of the data set is assigned to an idle node. Each node has two major computational tasks namely *map* and *reduce* functions computations. Depending upon the applications, a user can custom define the map and reduce functions. The abstract version of the map and reduce functions is:

Map function: The input for the map function is a (key,value) pair, and output is a list of (key,value) pairs. The formal representation of map function is of the form

$$\text{Map} : (k, v) \to \text{list}(k', v')$$

Reduce function: This function collects values by key, and then computes a value or a set of values depending upon the user definitions. The general representation of reduce function is of the form

$$\text{Reduce} : (k'', list(v'')) \to \text{list}(v'''),$$

where $k, k'$ and $k''$ are the keys, and $v, v', v''$ and $v'''$ are the values.

Several off-the-shelf implementations are available for the MapReduce procedure. The popular one is Hadoop, see (Apache Software Foundation, 2010). An interesting aspect of MapReduce approach is the system level issues like fixing the faults and distribution of data to the nodes are taken care of by the underlying implementation. The programmer needs to just define the map and reduce functions, see (Karloff et al., 2010; Mutharaju et al., 2010).

## 3.2 MapReduce for ABox Reasoning

In this section, we will explain in more detail how the completion rules given in Figure 1 can be reformulated to a format that is suitable for MapReduce framework. As explained in the previous section, the important computational tasks in MapReduce framework are computing map and reduce functions. Informally the MapReduce framework, in the context of completion rules for reasoning, works as follows: (a) based on the preconditions (ignoring the precondition which makes sure that the duplicate inferences are not computed), the map function identifies a key and (b) the reduce function, based on the key, completes the application of the rule. To illustrate this idea, we choose the $\sqsubseteq$- rule given in Figure 1. The preconditions in this rule are $C(a) \in \mathcal{A}^*$ and $C \sqsubseteq D \in \mathcal{T}$. The common concept among these preconditions is $C$, and $C$ can be used as a key. So the map function, in the case of $\sqsubseteq$- rule, computes the pairs $(C, C(a) \in \mathcal{A})$ and $(C, C \sqsubseteq D \in \mathcal{T})$.

| Expansion Rule | Key |
|---|---|
| $\sqcap^+_{aux}$-rule : if $C(a) \in \mathcal{A}^*$ and $C \sqcap D \in \mathcal{C}_{\Sigma,\mathbb{S}}$, then $\mathbb{T}_1 := \mathbb{T}_1 \cup \{(C \sqcap D, C(a))\}$; | $C$ |
| $\sqcap^+$-rule : if $D(a) \in \mathcal{A}^*$ and $(C \sqcap D, C(a)) \in \mathbb{T}_1$, then $\mathcal{A}^* := \mathcal{A}^* \cup \{C \sqcap D(a)\}$; | $D$ |
| $\sqcap^-$-rule : if $C \sqcap D(a) \in \mathcal{A}^*$, then $\mathcal{A}^* := \mathcal{A}^* \cup \{C(a), D(a)\}$; | $C \sqcap D$ |
| $\exists^+_{aux}$-rule : if $r(a,b) \in \mathcal{A}^*$ and $\exists r.C \in \mathcal{C}_{\Sigma,\mathbb{S}}$, then $\mathbb{T}_2 := \mathbb{T}_2 \cup \{(\exists r.C, r(a,b))\}$; | $r$ |
| $\exists^+$-rule : if $C(b) \in \mathcal{A}^*$ and $(\exists r.C, r(a,b)) \in \mathbb{T}_2$, then $\mathcal{A}^* := \mathcal{A}^* \cup \{\exists r.C(a)\}$; | $C$ |
| $\exists^-$-rule : if $\exists r.C(a) \in \mathcal{A}^*$, then $\mathcal{A}^* := \mathcal{A}^* \cup \{r(a, w^r_C), C(w^r_C)\}$, where $w^r_C \in \mathcal{W}$; | $\exists r.C$ |
| $\sqsubseteq$ -rule : if $C(a) \in \mathcal{A}^*$ and $C \sqsubseteq D \in \mathcal{T}$, then $\mathcal{A}^* := \mathcal{A}^* \cup \{D(a)\}$; | $C$ |
| $H$-rule : if $r(a,b) \in \mathcal{A}^*$ and $r \sqsubseteq s \in \mathcal{R}$, then $\mathcal{A}^* := \mathcal{A}^* \cup \{s(a,b)\}$. | $r$ |

Figure 2: Reformulated ABox expansion rules.

Now, the reduce function collects pairs with the same key and completes the application of $\sqsubseteq$- rule which means the assertion $D(a)$ is added to the set $\mathcal{A}^*$. It is straightforward to cast $\sqcap^-$-, $\exists^-$-, $\sqsubseteq$- and $H$- rules directly into MapReduce format. To modify $\sqcap^+$ and $\exists^+$- rules in MapReduce format, we use a two step method which is similar to the one used in (Mutharaju et al., 2010; Mutharaju, 2016). The $\sqcap^+$- rule displayed in Figure 1 is split into two rules namely $\sqcap^+_{aux}$ and $\sqcap^+$ which are given in Figure 2. $\sqcap^+_{aux}$-rule computes a set $\mathbb{T}_1$ whose elements are not elements of the set $\mathcal{A}^*$ which is the output of the reasoning algorithm. The purpose of computing the set $\mathbb{T}_1$ is to provide necessary information to the reduce function about the conjuncts and the object involved so that reduce function can compute conjunction of two concept assertions with the same object. Each element of the set $\mathbb{T}_1$ is a pair whose first part contains information about the conjuncts and the second part contains information about the object.

Similarly, the $\exists^+$- rule displayed in Figure 1 is split into two rules namely $\exists^+_{aux}$ and $\exists^+$ which are given in Figure 2. $\exists^+_{aux}$ computes a set $\mathbb{T}_2$ whose elements are not elements of the set $\mathcal{A}^*$. The elements of the set $\mathbb{T}_2$ provide information to the reduce function so that it computes a concept assertion involving 'there exists' constructor. Each element of the set $\mathbb{T}_2$ is a pair whose first part contains information about the concept and the second part contains information about the objects. The $\sqcap^+$- and $\exists^+$- rules given in Figure 2 compute the respective ABox assertions. The sets $\mathbb{T}_1$ and $\mathbb{T}_2$ are initialized as $\emptyset$. Note that the result of the completion rules $\sqcap^+_{aux}$ and $\exists^+_{aux}$ do not follow syntax of $\mathcal{ELH}$ language.

Each element of $\mathbb{T}_1$ is a pair of preconditions of $\sqcap^+_{aux}$- rule. This rule is of the form $p \wedge q \Rightarrow p \wedge q$, where $p$ and $q$ are propositional variables, which is a tautology. Therefore, $\sqcap^+_{aux}$- rule is sound. $\sqcap^+$- rule in Figure 2 simulates exactly the same result of the $\sqcap^+$- rule in Figure 1. The same argument holds true for the $\exists^+_{aux}$- and $\exists^+$- rules. The application of $\exists^-$- rule on the precondition $\exists r.C(a)$ would result in adding the assertions $r(a, w^r_C)$ and $C(w^r_C)$ into the set $\mathcal{A}^*$ even in the case that the assertions $r(a,b)$ and $C(b)$ already available in $\mathcal{A}^*$ for some witness $b \in \mathcal{O}_\Sigma$. Also $\exists^-$- rule is sound. The remaining all other rules in Figure 2 are same as the rules in Figure 1 except for the preconditions which guarantee no duplication in the output. The reformulated ABox reasoning algorithm given in Figure 2 terminates if no application of any of the rule given in Figure 2 is applicable. Every new addition of ABox assertion into $\mathcal{A}$ is entailed by $\Sigma$. Since the ABox reasoning algorithm in Figure 1 is sound, complete and terminate so as the algorithm in Figure 2. Hence the reformulated algorithm in Figure 2 is correct, and the worst case running time of this ABox algorithm is cubic polynomial in the size of input KB and the set $\mathcal{C}_{\Sigma,\mathbb{S}}$.

## 3.3 Parallelization using MapReduce for ABox Reasoning

In this section, we discuss how parallelization is implemented in ABox reasoning procedure using MapReduce approach. We consider the completion rules given in Figure 2 with its respective keys which are already reformulated in a way that they are suitable for implementation of parallelization method. Here the inputs are the KB $\Sigma = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, $\mathcal{C}_{\Sigma,\mathbb{S}}$, $\mathbb{T}_1$ and $\mathbb{T}_2$. The set $\mathcal{A}^*$ is initialized as $\mathcal{A}$ and is expanded using the following strategy: The completion rules are applied iteratively so that in a particular iteration, one rule is applied. In a given iteration, the elements of the sets $\mathcal{A}$, $\mathcal{T}$, $\mathcal{R}$, $C_{\Sigma,\mathbb{S}}$, $\mathbb{T}_1$ and $\mathbb{T}_2$ are partitioned into subsets. Each subset is distributed to different computing nodes. Each node computes first map function and then computes reduce function. Each map-reduce computational cycle results in the parallel application of one of the completion rules. In the map computational step, based on the rule chosen for the application, the elements from either the set $\mathcal{A}^*$ or from the sets $\mathcal{A}^*$ and $C_{\Sigma,\mathbb{S}}$ or from the sets $\mathcal{A}^*$ and $\mathcal{T}$ or from the sets $\mathcal{A}^*$ and $\mathcal{R}$ which satisfy any of the preconditions of the rule are selected and, the relevant (key,value) pairs are computed. Here, key is a concept or role name common to the preconditions of the rule and the value is the preconditions itself.

Algorithm 1: MapReduce algorithm for the rule $\sqcap^-$.

```
map(key,value)
    /* key: line number (not relevant)*/
    /* value: an assertion from ABox $\mathcal{A}^*$*/
if value == $C \sqcap D(a) \in \mathcal{A}^*$ then
    return $(C \sqcap D, C \sqcap D(a) \in \mathcal{A}^*)$
end if

reduce(key,values)
    /* key: $C \sqcap D$ (concept)*/
    /* values: assertions from ABox $\mathcal{A}^*$*/
for all $v$ in values do
    if $v == C \sqcap D(a) \in \mathcal{A}^*$ then
        return $C(a), D(a) \in \mathcal{A}^*$
    end if
end for
```

In the reduce computational step, all the pairs having the same key are collected from different nodes and the results of the completion rule are obtained. During this reduce computational step, all valid combinations of values are taken onto account. At the end of each iteration, all the duplicate assertions are removed from the set $\mathcal{A}^*$. Iterations (map-reduce cycle) are continued until no further new addition of assertions into $\mathcal{A}^*$. That is, the set $\mathcal{A}^*$ obtained at the end of any two consecutive iterations remains same. The strategy discussed here is similar to the strategies presented in (Mutharaju et al., 2010; Mutharaju, 2016).

Map and reduce functions of $\sqcap^-$ and $\sqsubseteq$ reformulated ABox completion rules given in Figure 2 are defined in Algorithm 1 and Algorithm 2 respectively. For the remaining rules in Figure 2, map and reduce functions can be defined in a similar way. Algorithm 1 describes map and reduce functions for the $\sqcap^-$-rule. The input for the map function is an element of $\mathcal{A}^*$. The output of the map function is a list of relevant (key,value) pairs. A list of values of same key is accepted by the reduce function. At this point of time, every possible combination of values is tested to make sure the $\sqcap^-$-rule is applicable. The output of the reduce function is a list of elements which is added to the set $\mathcal{A}^*$. Algorithm 2 describes map and reduce functions for the $\sqsubseteq$-rule. Since this algorithm is similar to Algorithm 1, the discussion on Algorithm 2 is omitted.

Algorithm 2: MapReduce algorithm for the rule $\sqsubseteq$.

```
map(key,value)
    /* key: line number (not relevant)*/
    /* value: an assertion from ABox $\mathcal{A}^*$ or an axiom from TBox $\mathcal{T}$ */
if value == $C(a) \in \mathcal{A}^*$ then
    return $(C, C(a) \in \mathcal{A}^*)$
else if value == $C \sqsubseteq D \in \mathcal{T}$ then
    return $(C, C \sqsubseteq D \in \mathcal{T})$
end if

reduce(key,values)
    /* key: $C$ (concept)*/
    /* values: assertions from ABox $\mathcal{A}^*$ or axioms from TBox $\mathcal{T}$ */
for all $v_1$ in values do
    for all $v_2$ in values do
        if $v_1 == C(a) \in \mathcal{A}^*$ and $v_2 == C \sqsubseteq D \in \mathcal{T}$ then
            return $D(a) \in \mathcal{A}^*$
        end if
    end for
end for
```

As indicated in (Mutharaju, 2016), the algorithm presented in this section can compute duplicate information. In each map-reduce computation cycle, duplicate information will be removed. Removal of the duplicates from the set $\mathcal{A}^*$ incurs additional computational cost which affects the performance of the algorithm. The performance of this algorithm can be optimized by using the distributed computational models reported in (Mutharaju, 2016; Urbani et al., 2010).

## 4 SECRECY-PRESERVING REASONING

Let $\Sigma = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ be an $\mathcal{ELH}$ KB. Also let $\mathbb{S} \subseteq \mathcal{A}^* \setminus AX$ be the "secrecy set" to be protected from the querying agent. Given $\Sigma$ and $\mathbb{S}$, the objective is to answer assertion queries while preserving secrecy. Our approach is to compute a set $\mathbb{E}$, where $\mathbb{S} \subseteq \mathbb{E} \subseteq \mathcal{A}^* \setminus AX$, called the *secrecy envelope* for $\mathbb{S}$, so that protecting $\mathbb{E}$ the querying agent cannot logically infer any assertion in $\mathbb{S}$, see (Tao et al., 2015). We briefly explain the role of OWA in answering the queries and how it helps protecting the secrets. When answering a query with “Unknown”, the querying agent should not be able to distinguish between the case that the answer to the query is truly unknown to the KB reasoner and the case that the answer is being protected for reasons of secrecy. We envision a situation in which once the ABox $\mathcal{A}^*$ is computed, a reasoner $\mathfrak{R}$ is associated with it. $\mathfrak{R}$ is designed to answer queries as follows: If a query cannot be inferred from $\Sigma$, the answer is “Unknown”. If it can be inferred and it is not in $\mathbb{E}$, the answer is “Yes”; otherwise, the answer is “Unknown”. Note that since the syntax of $\mathcal{ELH}$ does not include negation, an $\mathcal{ELH}$ KB cannot entail a negative query.

We make the following assumptions about the capabilities of the querying agent:

(a) It does not have direct access to the KB $\Sigma$, but is aware of the underlying vocabulary,
(b) It does not know about witness set $\mathcal{W}$,
(c) It can ask queries in the form of assertions, and
(d) It cannot ask queries in the form of general concept or role inclusions.

We formally define the notion of an envelope in the following:

**Definition 1.** *Let $\Sigma = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ be a $\mathcal{ELH}$ KB, and let $\mathbb{S}$ be a finite secrecy set. The secrecy envelope $\mathbb{E}$ of $\mathbb{S}$ have the following properties:*

- *$\mathbb{S} \subseteq \mathbb{E} \subseteq \mathcal{A}^* \setminus AX$, and*
- *for every $\alpha \in \mathbb{E}$, $\mathcal{A}^* \setminus \mathbb{E} \not\models \alpha$.*

The intuition for the above definition is that no information in $\mathbb{E}$ can be inferred from the set $\mathcal{A}^* \setminus \mathbb{E}$. To compute an envelope, we use the idea of inverting the rules of Figure 1 as given in (Tao et al., 2010; Tao et al., 2015). Induced by the ABox expansion rules in Figures 1, we define the corresponding “inverted” ABox expansion rules in Figure 3. These inverted expansion rules are denoted by prefixing Inv- to the name of the corresponding expansion rules. Note that the $\exists^-$–rule does not have its corresponding inverted rule. The reason for the ”omission” is that an application of this rule results in adding assertions with individual names from the witness set which the querying agent is barred from asking about.

The envelope $\mathbb{E}$ is computed by initializing it to $\mathbb{S}$ and then expanding it using the inverted expansion rules listed in Figure 3 until no further applications are possible. We denote by $\Lambda^{\mathbb{S}}$ the algorithm which computes the set $\mathbb{E}$. Due to non-determinism in applying the rules Inv-$\sqcap^+$ and Inv-$\exists^+$, different executions of $\Lambda^{\mathbb{S}}$ may output different envelopes. Since $\mathcal{A}^*$ is finite, the computation of $\Lambda^{\mathbb{S}}$ terminates. Let $\mathbb{E}$ be an output of $\Lambda^{\mathbb{S}}$. Since the size of $\mathcal{A}^*$ is cubic polynomial in $|\Sigma| + |\mathcal{C}_{\Sigma,\mathbb{S}}|$, and each application of inverted expansion rule moves some assertions from $\mathcal{A}^*$ into $\mathbb{E}$, the size of $\mathbb{E}$ is at most the size of $\mathcal{A}^*$. Therefore, to compute the envelope $\mathbb{E}$, $\Lambda^{\mathbb{S}}$ takes $O((|\Sigma| + |\mathcal{C}_{\Sigma,\mathbb{S}}|)^3)$. The proof of correctness of $\Lambda^{\mathbb{S}}$ is omitted.

| | |
|---|---|
| Inv-$\sqcap^+$-rule : | if $C \sqcap D(a) \in \mathbb{E}$, $C \sqcap D \in \mathcal{C}_{\Sigma,\mathbb{S}}$ and $\{C(a), D(a)\} \subseteq \mathcal{A}^* \setminus \mathbb{E}$, then $\mathbb{E} := \mathbb{E} \cup \{C(a)\}$ or $\mathbb{E} := \mathbb{E} \cup \{D(a)\}$; |
| Inv-$\sqcap^-$-rule : | if $\{C(a), D(a)\} \cap \mathbb{E} \neq \emptyset$ and $C \sqcap D(a) \in \mathcal{A}^* \setminus \mathbb{E}$, then $\mathbb{E} := \mathbb{E} \cup \{C \sqcap D(a)\}$; |
| Inv-$\exists^+$-rule : | if $\exists r.C(a) \in \mathbb{E}$, $\exists r.C \in \mathcal{C}_{\Sigma,\mathbb{S}}$ and $\{r(a,b), C(b)\} \subseteq \mathcal{A}^* \setminus \mathbb{E}$ with $b \in O^*$, then $\mathbb{E} := \mathbb{E} \cup \{r(a,b)\}$ or $\mathbb{E} := \mathbb{E} \cup \{C(b)\}$; |
| Inv-$\sqsubseteq$-rule : | if $D(a) \in \mathbb{E}$, $C \sqsubseteq D \in \mathcal{T}$ and $C(a) \in \mathcal{A}^* \setminus \mathbb{E}$, then $\mathbb{E} := \mathbb{E} \cup \{C(a)\}$; |
| Inv-$H$-rule : | if $s(a,b) \in \mathbb{E}$, $r \sqsubseteq s \in \mathcal{R}$ and $r(a,b) \in \mathcal{A}^* \setminus \mathbb{E}$, then $\mathbb{E} := \mathbb{E} \cup \{r(a,b)\}$. |

Figure 3: Inverted Tableau expansion rules.

## 4.1 MapReduce for Computing an Envelope $\mathbb{E}$

An important step in computing a secrecy envelope for a given secrecy set using the MapReduce framework is to cast the inverted ABox tableau expansion rules given in Figure 3 to a format that is suitable for MapReduce procedure. Since a detailed discussion of various computational tasks in MapReduce framework in the context of completion rules for reasoning is given in section 3.2, we present a short explanation about the modified inverted rules given in Figure 4. To modify each rule in Figure 3 except for the rule Inv-$\sqcap^-$ in MapReduce format, we use a two step method as explained in detail in section 3.2. The Inv-$\sqcap^+$- rule displayed in Figure 3 is split into two rules namely Inv-$\sqcap^+_{aux}$ and Inv-$\sqcap^+$ given in Figure 4. Inv-$\sqcap^+_{aux}$-rule computes a set $\mathbb{T}_3$ whose elements are not elements of the set $\mathbb{E}$. The purpose of computing the set $\mathbb{T}_3$ is to provide necessary information to the reduce function of the rule Inv-$\sqcap^+$. Using this information, the reduce function computes an assertion that goes in to $\mathbb{E}$ corresponding to a secret involving conjunction constructor. Otherwise, that secret may be revealed.

Similarly, each inverted expansion rules Inv-$\exists^+$, Inv-$\sqsubseteq$ and Inv-$H$ displayed in Figure 3 is split into a pair of rules. The first rule in each of these pairs of rules is an auxiliary rule Inv-$\exists^+_{aux}$, Inv-$\sqsubseteq_{aux}$ and Inv-$H_{aux}$ which computes respectively a set $\mathbb{T}_4$, $\mathbb{T}_5$ and $\mathbb{T}_6$. As explained in the previous paragraph, each of these sets provide information to the reduced function

| | Inverted Expansion Rule | Key |
|---|---|---|
| Inv-$\sqcap^+_{aux}$ : | if $C(a) \in \mathcal{A}^* \setminus \mathbb{E}$ and $C \sqcap D(a) \in \mathbb{E}$, then $\mathbb{T}_3 := \mathbb{T}_3 \cup \{(C \sqcap D(a), C(a))\}$; | $C$ |
| Inv-$\sqcap^+$ : | if $D(a) \in \mathcal{A}^* \setminus \mathbb{E}$ and $(C \sqcap D(a), C(a)) \in \mathbb{T}_3$, then $\mathbb{E} := \mathbb{E} \cup \{C(a)\}$ or $\mathbb{E} := \mathbb{E} \cup \{D(a)\}$; | $D$ |
| Inv-$\sqcap^-_L$ : | if $C \sqcap D(a) \in \mathcal{A}^* \setminus \mathbb{E}$ and $C(a) \in \mathbb{E}$, then $\mathbb{E} := \mathbb{E} \cup \{C \sqcap D(a)\}$; | $C$ |
| Inv-$\sqcap^-_R$ : | if $C \sqcap D(a) \in \mathcal{A}^* \setminus \mathbb{E}$ and $D(a) \in \mathbb{E}$, then $\mathbb{E} := \mathbb{E} \cup \{C \sqcap D(a)\}$; | $D$ |
| Inv-$\exists^+_{aux}$ : | if $r(a,b) \in \mathcal{A}^* \setminus \mathbb{E}$ and $\exists r.C(a) \in \mathbb{E}$, then $\mathbb{T}_4 := \mathbb{T}_4 \cup \{(\exists r.C(a), r(a,b))\}$; | $r$ |
| Inv-$\exists^+$ : | if $C(b) \in \mathcal{A}^* \setminus \mathbb{E}$ and $(\exists r.C(a), r(a,b)) \in \mathbb{T}_4$, then $\mathbb{E} := \mathbb{E} \cup \{r(a,b)\}$ or $\mathbb{E} := \mathbb{E} \cup \{C(b)\}$; | $C$ |
| Inv-$\sqsubseteq_{aux}$: | if $C(a) \in \mathcal{A}^* \setminus \mathbb{E}$ and $C \sqsubseteq D \in \mathcal{T}$, then $\mathbb{T}_5 := \mathbb{T}_5 \cup \{(C \sqsubseteq D, C(a))\}$; | $C$ |
| Inv-$\sqsubseteq$: | if $D(a) \in \mathbb{E}$ and $(C \sqsubseteq D, C(a)) \in \mathbb{T}_5$, then $\mathbb{E} := \mathbb{E} \cup \{C(a)\}$; | $D$ |
| Inv-$H_{aux}$ : | if $r(a,b) \in \mathcal{A}^* \setminus \mathbb{E}$ and $r \sqsubseteq s \in \mathcal{R}$, then $\mathbb{T}_6 := \mathbb{T}_6 \cup \{(r \sqsubseteq s, r(a,b))\}$; | $r$ |
| Inv-$H$ : | if $s(a,b) \in \mathbb{E}$ and $(r \sqsubseteq s, r(a,b)) \in \mathbb{T}_6$, then $\mathbb{E} := \mathbb{E} \cup \{r(a,b)\}$. | $s$ |

Figure 4: Reformulated Inverted Tableau expansion rules.

of second rule in the respective pairs. The Inv-$\sqcap^+$, Inv-$\exists^+$, Inv-$\sqsubseteq$ and Inv-$H$- rules given in Figure 4 compute assertions that belong to set $\mathbb{E}$. The sets $\mathbb{T}_3$, $\mathbb{T}_4$, $\mathbb{T}_5$ and $\mathbb{T}_6$ are initialized as $\emptyset$. Note that the result of the completion rules $\sqcap^+_{aux}$, $\exists^+_{aux}$, $\sqsubseteq_{aux}$ and $H_{aux}$ do not follow syntax of $\mathcal{ELH}$ language.

The intuition behind the rule Inv-$\sqcap^-$ given in Figure 3 is that if either $C(a)$ or $D(a)$ is in $\mathbb{E}$, then $C \sqcap D(a)$ should be in $\mathbb{E}$. That is, to protect the secret information $C(a)$ or $D(a)$, we need to add $C \sqcap D(a)$ into the set $\mathbb{E}$. To cast this rule in MapReduce format, we split it into two rules Inv-$\sqcap^-_L$ and Inv-$\sqcap^-_R$. Each rule computes the same assertion involving conjunction constructor. Since at the end of each map-reduce cycle, the redundant information are removed, the final output $\mathbb{E}$ of the tableau algorithm given in Figure 4 is free from duplicate assertions.

As discussed in section 3.2, all the auxiliary rules in Figure 4 are of the form $p \wedge q \Rightarrow p \wedge q$, where $p$ and $q$ are propositional variables, which is a tautology. Therefore, all the auxillary rules are sound. The remaining rules in Figure 4 simulate exactly the same result of the rules in Figure 3. Since the tableau algorithm given in Figure 3 is correct so as the algorithm given in Figure 4. Hence the reformulated inverted tableau algorithm in Figure 4 is correct,

## 4.2 Parallelization using MapReduce for Envelope Computation

A brief account of design and implementation of map and reduce functions for each reformulated inverted completion rules given in Figure 4 is discussed in this section, for more details see section 3.3. We consider the inverted expansion rules with its respective keys which are already reformulated in a way that is suitable for implementation of parallelization method. The inputs are the KB $\Sigma = \langle \mathcal{A}^*, \mathcal{T}, \mathcal{R} \rangle$, $C_{\Sigma,\mathbb{S}}$, $\mathbb{S}$, $\mathbb{T}_3$, $\mathbb{T}_4$. $\mathbb{T}_5$ and $\mathbb{T}_6$. The set $\mathbb{E}$ is initialized as $\mathbb{S}$ and is expanded using the following strategy. The completion rules are applied iteratively so that in a particular iteration, one rule is applied. In a given iteration, the elements of the sets $\mathcal{A}^*$, $\mathcal{T}$, $\mathcal{R}$, $C_{\Sigma,\mathbb{S}}$, $\mathbb{S}$, $\mathbb{T}_3$, $\mathbb{T}_4$, $\mathbb{T}_5$ and $\mathbb{T}_6$ are partitioned into subsets. Each subset is distributed to different computing nodes. Each node computes first map function and then computes reduce function. Each map-reduce computational cycle results in the parallel application of one of the rules. At the end of each map-reduce cycle, all the duplicate assertions are removed from the set $\mathbb{E}$. Map-reduce cycle are continued until no further new addition of assertions into $\mathbb{E}$. That is, the set $\mathbb{E}$ obtained at the end of any two consecutive iterations remains same.

Algorithm 3: MapReduce algorithm for Inv-$\sqcap_L^-$.

```
map(key,value)
    /* key: line number (not relevant)*/
    /* value: an assertion from ABox $\mathcal{A}^* \setminus \mathbb{E}$ or an assertion from $\mathbb{E}$ */
if value == $C \sqcap D(a) \in \mathcal{A}^* \setminus \mathbb{E}$ then
    return ($C, C \sqcap D(a) \in \mathcal{A}^* \setminus \mathbb{E}$)
else if value == $C(a) \in \mathbb{E}$ then
    return ($C, C(a)$)
end if

reduce(key,values)
    /* key: $C$ (concept)*/
    /* values: assertions from ABox $\mathcal{A}^* \setminus \mathbb{E}$ or assertions from the set $\mathbb{E}$*/
for all $v_1$ in values do
    for all $v_2$ in values do
        if $v_1 == C \sqcap D(a) \in \mathcal{A}^* \setminus \mathbb{E}$ and $v_2 == C(a)$ then
            return $C \sqcap D(a) \in \mathbb{E}$
        end if
    end for
end for
```

Algorithm 4: MapReduce algorithm for Inv-$\sqsubseteq_{aux}$.

```
map(key,value)
    /* key: line number (not relevant)*/
    /* value: an assertion from ABox $\mathcal{A}^* \setminus \mathbb{E}$ or an axiom from TBox $\mathcal{T}$ */
if value == $C(a) \in \mathcal{A}^* \setminus \mathbb{E}$ then
    return ($C, C(a) \in \mathcal{A}^* \setminus \mathbb{E}$)
else if value == $C \sqsubseteq D \in \mathcal{T}$ then
    return ($C, C \sqsubseteq D$)
end if
reduce(key,values)
    /* key: $C$ (concept)*/
    /* values: assertions from ABox $\mathcal{A}^*$ or axioms from TBox $\mathcal{T}$ */
for all $v_1$ in values do
    for all $v_2$ in values do
        if $v_1 == C(a) \in \mathcal{A}^* \setminus \mathbb{E}$ and $v_2 == C \sqsubseteq D$ then
            return ($C \sqsubseteq D, C(a)) \in \mathbb{T}_5$
        end if
    end for
end for
```

Map and reduce functions for Inv-$\sqcap_L^-$, Inv-$\sqsubseteq_{aux}$ and Inv-$\sqsubseteq$ reformulated inverted completion rules given in Figure 4 are defined in Algorithm 3 through Algorithm 5. For the remaining rules in Figure 4, map and reduce functions can be defined in a similar way. Algorithm 3 describes map and reduce functions for the $\sqcap_L^-$-rule. The input for the map function is an element of $\mathcal{A}^* \setminus \mathbb{E}$ and an element in $\mathbb{E}$. The output of

Algorithm 5: MapReduce algorithm for Inv-$\sqsubseteq$.

```
map(key,value)
    /* key: line number (not relevant)*/
    /* value: an assertion in $\mathbb{E}$ or an element in the set $\mathbb{T}_5$ */
if value == $D(a) \in \mathbb{E}$ then
    return ($D, D(a) \in \mathbb{E}$)
else if value == ($C \sqsubseteq D, C(a)) \in \mathbb{T}_5$ then
    return ($D, (C \sqsubseteq D, C(a))$)
end if
reduce(key,values)
    /* key: $C$ (concept)*/
    /* values: assertions from the set $\mathbb{E}$ or elements from the set $\mathbb{T}_5$ */
for all $v_1$ in values do
    for all $v_2$ in values do
        if $v_1 == D(a) \in \mathbb{E}$ and $v_2 == (C \sqsubseteq D, C(a))$ then
            return $C(a) \in \mathbb{E}$
        end if
    end for
end for
```

the map function is a list of relevant (key,value) pairs. A list of values of same key is accepted by the reduce function. At this point of time, every possible combination of values is tested to make sure the $\sqcap_L^-$-rule is applicable. The output of the reduce function is a list of elements which is added to the set $\mathbb{E}$. The map and reduce functions of remaining algorithms are similar to Algorithm 3 and can be explained in the same way. As discussed in section 3.3, the procedure that computes envelope can output duplicate information. At the end of each map-reduce computation cycle, duplicate assertions will be removed. Removal of the duplicates from the set $\mathbb{E}$ incurs additional computational cost which affects the performance of this computational procedure. The performance of this procedure can be improved by using the distributed computational models reported in (Mutharaju, 2016; Urbani et al., 2010).

## 5 SUMMARY

In this paper we have studied the problem of secrecy-preserving reasoning in $\mathcal{ELH}$ KBs using MapReduce framework. Let $\Sigma = \langle \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ be an $\mathcal{ELH}$ KB and assume that the size of the set $\mathcal{A}$ is very large. Our main contribution in this paper is to use MapReduce procedure within reasoning algorithms to compute a finite set of assertional consequences $\mathcal{A}^*$ and an envelope $\mathbb{E}$ which is super set of the secrecy set $\mathbb{S}$. To the best of our knowledge, secrecy-preserving reasoning with MapReduce framework has not been studied before. For the query answering part, we assume that the sets $\mathcal{A}^*$ and $\mathbb{E}$ are precomputed. The set $\mathcal{A}^* \setminus \mathbb{E}$ is free from all the secret information, and no secret information can be inferred from it. The queries from the querying agent are answered based on the information available in the set $\mathcal{A}^* \setminus \mathbb{E}$. Note that $\mathcal{A}^* \setminus \mathbb{E}$ is finite and does not contain answer for all the non-confidential queries. A recursive query answering procedure is used to answer the non-consequential queries. For this purpose,we adopt the recursive query answering procedure reported in (Sivaprakasam and Slutzki, 2017) with the necessary changes. Our main emphasis in this paper is how to use MapReduce framework within the reasoning procedures to study the secrecy-preserving reasoning problem. The implementation of this query answering procedure in Hadoop tool (Apache Software Foundation, 2010) will be considered in our future work. Further, we will implement SPQA framework for a single querying agent in Protege (Protege - Stanford University, 1999), a knowledge representation and reasoning tool for $\mathcal{ELH}$ KBs. To study the performance of implementation of SPQA framework for single querying agent using MapReduce procedure, we will conduct experiments in very large ontologies SNOMED CT and GALEN (Dentler et al., 2011) in both Protege and Hadoop tools and compare their performances.

## REFERENCES

Apache Software Foundation (2010). http://hadoop.apache.org.

Bellomarini, L., Gottlob, G., Pieris, A., and Sallinger, E. (2018). Swift logic for big data and knowledge graphs. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 3–16. Springer.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Dentler, K., Cornet, R., Ten Teije, A., and De Keizer, N. (2011). Comparison of reasoners for large ontologies in the owl 2 el profile. *Semantic Web*, 2(2):71–87.

Hitzler, P. and Janowicz, K. (2013). Linked data, big data, and the 4th paradigm. *Semantic Web*, 4(3):233–235.

Karloff, H., Suri, S., and Vassilvitskii, S. (2010). A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948. SIAM.

Konstantopoulos, S., Charalambidis, A., Mouchakis, G., Troumpoukis, A., Jakobitsch, J., and Karkaletsis, V. (2016). Semantic web technologies and big data infrastructures: Sparql federated querying of heterogeneous big data stores. In *International Semantic Web Conference (Posters & Demos)*.

Krötzsch, M. (2012). Owl 2 profiles: An introduction to lightweight ontology languages. In *Reasoning Web International Summer School*, pages 112–183. Springer.

Möller, R., Neuenstadt, C., Özçep, Ö. L., and Wandelt, S. (2013). Advances in accessing big data with expressive ontologies. In *Annual Conference on Artificial Intelligence*, pages 118–129. Springer.

Mutharaju, R. (2016). *Distributed rule-based ontology reasoning*. PhD thesis, Wright State University.

Mutharaju, R., Maier, F., and Hitzler, P. (2010). A mapreduce algorithm for $\mathcal{EL}^+$. In *23rd International Workshop on Description Logics DL2010*, volume 456.

Protege - Stanford University (1999). http://protege.stanford.edu.

Reinsel, D., Gantz, J., and Rydning, J. (2017). Data age 2025.

Sivaprakasam, G. K. and Slutzki, G. (2016). Secrecy-preserving query answering in ELH knowledge bases. In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016), Volume 2, Rome, Italy, February 24-26.*, pages 149–159.

Sivaprakasam, G. K. and Slutzki, G. (2017). Keeping secrets in modalized DL knowledge bases. In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence, ICAART 2017, Volume 2, Porto, Portugal, February 24-26.*, pages 591–598.

Tao, J., Slutzki, G., and Honavar, V. (2010). Secrecy-preserving query answering for instance checking in $\mathcal{EL}$. In *International Conference on Web Reasoning and Rule Systems*, pages 195–203. Springer.

Tao, J., Slutzki, G., and Honavar, V. (2015). A conceptual framework for secrecy-preserving reasoning in knowledge bases. *ACM Transactions on Computational Logic (TOCL)*, 16(1):3.

Urbani, J., Kotoulas, S., Maassen, J., Van Harmelen, F., and Bal, H. (2010). Owl reasoning with webpie: calculating the closure of 100 billion triples. In *Extended Semantic Web Conference*, pages 213–227. Springer.

Zhou, Y., Cuenca Grau, B., Horrocks, I., Wu, Z., and Banerjee, J. (2013). Making the most of your triple store: query answering in owl 2 using an rl reasoner. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1569–1580. ACM.