Parallel ABox Reasoning of $\mathcal{EL}$ Ontologies

Yuan Ren[1], Jeff Z. Pan[1] and Kevin Lee[2]

[1]University of Aberdeen, Aberdeen, UK
[2]NICTA, Australia

Abstract. In order to support the vision of the Semantic Web, ontology reasoning needs to be highly scalable and efficient. A natural way to achieve scalability and efficiency is to develop parallel ABox reasoning algorithms for tractable OWL 2 profiles to distribute the load between different computation units within a reasoning system. So far there have been some work on parallel ABox reasoning algorithms for the pD* fragment of OWL 2 RL. However, there is still no work on parallel ABox reasoning algorithm for OWL 2 EL, which is the language for many influential ontologies (such as the SNOMED CT ontology). In this paper, we extend a parallel TBox reasoning algorithm [5] for $\mathcal{ELH}_{\mathcal{R}+}$ to parallel ABox reasoning algorithms for $\mathcal{ELH}_{\bot,\mathcal{R}+}$, which also supports the bottom concept so as to model disjointness and inconsistency. In design of algorithms, we exploit the characteristic of ABox reasonings to improve parallelisation and reduce unnecessary resource cost. Our evaluation shows that a naive implementation of our approach can compute all ABox entailments of a Not-Galen$^-$ ontology with about 1 million individuals and 9 million axioms in about 3 minutes.

1 Introduction

Ontologies are the knowledge infrastructures of the Semantic Web and many intelligent systems. In order to support the vision of the Semantic Web, ontology reasoning services need to be highly scalable and efficient. The modern ontology language standard, the W3C OWL Recommendation, is based on (different) description logics (DLs). In the last decades, DL reasoning technologies have been developed to support inference with ontologies. Well-optimised DL reasoning systems, such as FaCT++[1], HermiT[2], Pellet[3], CEL[4], CB[5] and TrOWL[6], have been implemented with different reasoning technologies. So far, these systems are designed for a single computation core. Reasoning is performed sequentially and can not be parallelised.

A natural way to achieve scalability and efficiency is to develop parallel ABox reasoning algorithms for tractable OWL 2 profiles, such as OWL 2 EL and OWL 2 RL, that can distribute the load between different computation units within a reasoning system:

[1] http://owl.man.ac.uk/factplusplus/
[2] http://hermit-reasoner.com/
[3] http://clarkparsia.com/pellet/
[4] http://lat.inf.tu-dresden.de/systems/cel/
[5] http://code.google.com/p/cb-reasoner/
[6] http://trowl.eu/

One direction is to perform parallel reasoning with a cluster of multiple computer nodes (or simply, *peers*). In Marvin [9], peers use a divide-conquer-swap strategy for RDFS inference. Weaver and Handler propose a parallel RDFS inference engine [18]; peers use an ABox partitioning approach to RDFS inference. In SAOR [4], peers use optimised template rules for join-free inference in pD* [15]. In DRAGO [12], peers performs OWL DL reasoning under the setting of Distributed Description Logics [2], which support local reasoning at the price of sacrificing expressiveness in the links between local models. A distributed resolution algorithm for $\mathcal{ALC}$ was proposed in [10]. Different from the aforementioned work where each peer has the same capability, in this algorithm each peer is responsible for inferences for different types of literals, making certain peer(s) become potential bottleneck and a single-point-of-failure. This issue is addressed by the authors when they extend their algorithm for $\mathcal{ALCHIQ}$ [11]. MapReduce [3] has also been adopted to support ABox reasoning in RDFS [17], pD* [16] as well as justifications in pD* [19], and TBox reasoning in $\mathcal{EL}^+$ [7] (there is no implementation for the $\mathcal{EL}^+$ case yet).

Another direction is to perform parallel reasoning with multiple computation cores (or simply, *workers*) in a single computer. Soma and Prasanna [13] propose to use data-partitioning and rule-partitioning in their parallel algorithms for pD*. Liebig and M$\ddot{u}$ller [6] exploit the non-determinism introduced by disjunctions or number restrictions in the $\mathcal{SHN}$ tableau algorithm so that multiple workers can apply expansion rules on independent alternatives. Similarly, Meissner [8] proposes parallel expansions of independent branchings in an $\mathcal{ALC}$ tableau and experimented with 3 different strategies. Aslani and Haarslev [1] propose a parallel algorithm for OWL DL classification. Recently, Kazakov et al. [5] presented a lock-free parallel completion-based TBox classification algorithm for $\mathcal{ELH}_{\mathcal{R}+}$.

As discussed above, there have been work on parallel ABox reasoning for the pD* fragment of OWL 2 RL. However, there is still no work on parallel ABox reasoning algorithm for OWL 2 EL, in which many influential ontologies (such as the SNOMED CT ontology) are written. In this paper, we extend a parallel TBox reasoning algorithm [5] for $\mathcal{ELH}_{\mathcal{R}+}$ to a parallel and lock-free ABox reasoning algorithm for $\mathcal{ELH}_{\bot,\mathcal{R}+}$, which also supports the bottom concept so as to model disjointness and inconsistency. We exploit the different characteristic of ABox reasoning from TBox reasoning and optimise the design of completion rules and algorithms accordingly to improve parallelisation and reduce unnecessary resource cost. Particularly, we parallelise the initialisation of algorithms, separate TBox and ABox saturation, and streamline the processing of each axiom in each worker. Our evaluation shows that a naive implementation of our approach can handle combined complex TBox and large ABox efficiently.

The remainder of the paper is organised as follows: In Sec. 2 we introduce background knowledge of DLs $\mathcal{ELH}_{\mathcal{R}+}$ and $\mathcal{ELH}_{\bot,\mathcal{R}+}$, and the parallel $\mathcal{ELH}_{\mathcal{R}+}$ TBox classification algorithm in [5]. In Sec. 3 we explain the technical challenges , before presenting the completion rules and parallel ABox reasoning algorithms for $\mathcal{ELH}_{\bot,\mathcal{R}+}$ in Sec. 4. We present an implementation of our approach and our evaluation in Sec.5, before we conclude the paper in Sec. 6.

The proof of all lemmas and theorems are included in our online tech report at http://www.box.net/shared/mpqxgxydhhl2bpuus5f7.

2 Preliminary

2.1 The $\mathcal{ELH}_{\mathcal{R}+}$ and $\mathcal{ELH}_{\bot,\mathcal{R}+}$ DLs

A *signature* of an ontology $\mathcal{O}$ is a triple $\Sigma_{\mathcal{O}} = (\mathcal{CN}_{\mathcal{O}}, \mathcal{RN}_{\mathcal{O}}, \mathcal{IN}_{\mathcal{O}})$ consisting of three mutually disjoint finite sets of atomic concepts $\mathcal{CN}_{\mathcal{O}}$, atomic roles $\mathcal{RN}_{\mathcal{O}}$ and individuals $\mathcal{IN}_{\mathcal{O}}$. Given a signature, complex concepts in $\mathcal{ELH}_{\bot,\mathcal{R}+}$ can be defined inductively using the $\mathcal{ELH}_{\bot,\mathcal{R}+}$ constructors as in Table 1. $\mathcal{ELH}_{\mathcal{R}+}$ supports all $\mathcal{ELH}_{\bot,\mathcal{R}+}$ constructors except $\bot$. Two concepts C and D are equivalent if they mutually include each other, denoted by $C \equiv D$. An ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T}$ and an ABox

Table 1. $\mathcal{ELH}_{\bot,\mathcal{R}+}$ Syntax and Semantics

Concepts:		
atomic concept	A	$A^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{x \mid \exists y.\langle x, y\rangle \in r^{\mathcal{I}}$ and $y \in C^{\mathcal{I}}\}$
Roles:		
atomic role	r	$r^{\mathcal{I}}$
TBox Axioms:		
general concept inclusion (GCI):	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
role inclusion (RI):	$r \sqsubseteq s$	$r^{\mathcal{I}} \sqsubseteq s^{\mathcal{I}}$
role transitivity:	$Trans(t)$	$t^{\mathcal{I}} \times t^{\mathcal{I}} \subseteq t^{\mathcal{I}}$
ABox Axioms:		
class assertion:	$A(a)$	$a^{\mathcal{I}} \in A^{\mathcal{I}}$
role assertion:	$r(a, b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in r^{\mathcal{I}}$
individual equality:	$a \doteq b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
individual inequality:	$a \dot{\neq} b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$

$\mathcal{A}$, which are finite sets of TBox axioms and ABox axioms, respectively. $\mathcal{ELH}_{\bot,\mathcal{R}+}$ allows all axioms listed in Table 1. $\mathcal{ELH}_{\mathcal{R}+}$ allows all except individual inequalities.

An *interpretation* $\mathcal{I}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set and $\cdot^{\mathcal{I}}$ is a function that maps each atomic concept A to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each atomic role r to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and each individual a to an object $a^{I} \in \Delta^{\mathcal{I}}$. Interpretation function $\cdot^{\mathcal{I}}$ can be extended to complex concept as shown in Table 1.

An interpretation $\mathcal{I}$ is a model of an ontology $\mathcal{O}$, written $\mathcal{I} \models \mathcal{O}$, if it satisfies all axioms of $\mathcal{O}$ as shown in the lower part of Table 1. An axiom α is entailed by an ontology $\mathcal{O}$, written $\mathcal{O} \models \alpha$, iff all models of $\mathcal{O}$ satisfy α. A concept C is *satisfiable* w.r.t. an ontology $\mathcal{O}$ if there exists some model $\mathcal{I}$ of $\mathcal{O}$ such that $C^{\mathcal{I}} \neq \emptyset$. Given an ontology $\mathcal{O}$, we use $\sqsubseteq^*_{\mathcal{O}}$ to represent the relfexive transitive closure of RIs. It is easy to see that in an $\mathcal{ELH}_{\mathcal{R}+}$/ $\mathcal{ELH}_{\bot,\mathcal{R}+}$ ontology, all of such $\sqsubseteq^*_{\mathcal{O}}$ relations can be computed in polynomial time w.r.t. the size of $\mathcal{O}$.

In ABox reasoning, we are particularly interested in finding all atomic types and relations of all individuals, i.e. finding all $A(a)$ s.t. $a \in \mathcal{IN}_{\mathcal{O}}$, $A \in \mathcal{CN}_{\mathcal{O}}$, $\mathcal{O} \models A(a)$

and all $r(a,b)$ s.t. $a,b \in \mathcal{IN}_\mathcal{O}$, $r \in \mathcal{RN}_\mathcal{O}$ and $\mathcal{O} \models r(a,b)$. We call such a reasoning task *ABox classification*. Computing and maintaining ABox classification results can be very useful for efficient on-line instance retrieval and/or query answering.

2.2 Parallel TBox Classification of $\mathcal{ELH}_{\mathcal{R}+}$ Ontologies

Given an ontology $\mathcal{O}$, TBox classification is a reasoning task that computes all inclusions over atomic concepts in $\mathcal{O}$. Kazakov et. al [5] proposed an approach to parallel TBox classification for $\mathcal{ELH}_{\mathcal{R}+}$. They devise a set of completion rules as follows, where $D \to E$ is used to denote the special form of GCIs where D and E are both existential restrictions. Given an $\mathcal{ELH}_{\mathcal{R}+}$ ontology $\mathcal{O}$ that has no ABox, these rules infer $C \sqsubseteq D$ iff $\mathcal{O} \models C \sqsubseteq D$ for all C and D such that $C \sqsubseteq C \in$ **S** and D occurs in $\mathcal{O}$ (Theorem 1 of [5]), where **S** is the set of axioms closed under the following inference rules.

$$\mathbf{R}_\sqsubseteq \frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in \mathcal{O}$$
$$\mathbf{R}_\sqcap^- \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1; C \sqsubseteq D_2}$$
$$\mathbf{R}_\exists^- \frac{C \sqsubseteq \exists R.D}{D \sqsubseteq D}$$
$$\mathbf{R}_\top^+ \frac{C \sqsubseteq C}{C \sqsubseteq \top} : \top \text{ occurs in } \mathcal{O}$$
$$\mathbf{R}_\sqcap^+ \frac{C \sqsubseteq D_1, C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \text{ occurs in } \mathcal{O}$$
$$\mathbf{R}_\exists^+ \frac{C \sqsubseteq D}{\exists s.C \to \exists s.D} : \exists s.D \text{ occurs in } \mathcal{O}$$
$$\mathbf{R}_\mathcal{H} \frac{D \sqsubseteq \exists r.C,\ \exists s.C \to E}{D \sqsubseteq E} : r \sqsubseteq_\mathcal{O}^* s$$
$$\mathbf{R}_T \frac{D \sqsubseteq \exists r.C,\ \exists s.C \to E}{\exists t.D \to E} : r \sqsubseteq_\mathcal{O}^* t \sqsubseteq_\mathcal{O}^* s,\ Trans(t) \in \mathcal{O}$$

The completion rules are designed in a way that *all premises of each rule have a common concept* (the concept C in each rule), which is called a *context* of the corresponding premise axioms. Each context maintains a queue of axioms called *scheduled*, on which some completion rule can be applied, and a set of axioms called *processed*, on which some completion rule has already been applied. *An axiom can only be included in the scheduled queues and/or processed sets of its own contexts.* To ensure that multiple workers can share the queues and sets without locking them, they further devised a concurrency mechanism in which: (i) each worker will process a single context at a time and vice versa; (ii) the processing of all axioms in the scheduled queue of a context requires no axioms from the processed sets of other contexts. To realise all these, all contexts with non-empty schedules are arranged in a queue called *activeContexts*. A context can be added into the *activeContexts* queue only if it is not already in the queue.

Here are the key steps of the parallel TBox algorithm:

1. Tautology axiom $A \sqsubseteq A$ for each $A \in \mathcal{CN}_{\mathcal{O}}$ is added to the scheduled queues of A. All active contexts are added into the queue of *activeContexts*.
2. Every idle worker always looks for the next context in the *activeContexts* queue and processes axioms in its scheduled queue.
   (a) Pop an axiom from the scheduled queue, add it into the processed set of the context.
   (b) Apply completion rules to derive conclusions.
   (c) Add each derived conclusion into the scheduled queue of its corresponding contexts, which will be activated if possible.

Before we extend the parallel TBox reasoning algorithm to support ABox reasoning in Sec. 4, we first discuss the challenges to dealt with in parallel ABox reasoning.

3 Technical Challenges: Parallel ABox Reasoning

When we design parallel ABox classification algorithms, we need to consider the characteristic of ABox classification that distinguish it from TBox classification — the number of individuals is often much larger than the number of concepts and roles.

A naive way of doing ABox classification is to internalise the entire ABox into TBox (i.e., by converting assertions of the form $C(a)$ into $\{a\} \sqsubseteq C$ and $R(a, b)$ into $\{a\} \sqsubseteq \exists R.\{b\}$) and treat the internalised "nominals" as ordinary atomic concepts with the TBox classification algorithm. This is inefficient due to redundant computations. For example, axiom $\{a\} \sqsubseteq \exists r.C$ has two contexts $\{a\}$ and C. Thus this axiom will be added into the *scheduled* queues of both $\{a\}$ and C. In context C, this axiom will be saved into the *processed* set of C and further retrived as the left premise of Rule $\mathbf{R}_{\mathcal{H}}$ and/or $\mathbf{R}_{T}$, for some future right premise $\exists s.C \to E$. However our target language $\mathcal{ELH}_{\perp,\mathcal{R}+}$ does not support nominals. Therefore it is unnecessary to maintain $\{a\} \sqsubseteq \exists r.C$ in context C because any corresponding right premise $\exists s.C \to E$ will not contain any nominal, hence it can always be computed independently from (or before) the derivation of $\{a\} \sqsubseteq \exists r.C$. This provides means to optimise reasoning because the concept hierarchies and RI closures will be static when doing ABox reasoning (cf. Sec 4.2).

Furthermore, it is important to optimise the seemingly trivial parts, which could become non-trivial due to the large number of individuals, of the algorithm in order to speed up the reasoning. Particularly:

1. Instead of initialising the contexts in a sequential manner one should parallelise this process in order to gain further efficiency (cf. Sec 4.3).
2. When applying completion rules to derive conclusions, as described by steps 2 at the end of the last section, a reasoner usually needs to check the forms of input axioms, decide the applicable rules, check the forms of conclusion axioms, etc. Some of the checking could be skipped, as they are all dependent thus can be streamlined (cf. Sec 4.4).
3. After derivation, the conclusions are maintained in a set, and then immediately retrieved to get contexts. All retrieved contexts are also maintained in a set, and then immediately retrieved for activation. One should be able to skip such save/retrieve steps and directly use the conclusions and their contexts given that the forms of conclusions are known to the reasoner (cf. Sec 4.4).

4. When an axiom is added into the *scheduled* queue of a context, the worker needs to activate this context for further processing. However if the context is the context currently under processing of the worker, such activation can be skipped. We only need to activate a context if we do not know it is the same as the current context (cf. Sec 4.4).

4 Approach

In this section we present parallel TBox and ABox classification algorithms for $\mathcal{ELH}_{\bot,\mathcal{R}+}$. We first present the new completion rules and then the algorithms.

4.1 TBox Completion Rules

We first extend the $\mathcal{ELH}_{\mathcal{R}+}$ TBox completion rules to support the bottom concept with the following rule for the $\bot$ concept:

$$\mathbf{R}_{\bot}\,\frac{D \sqsubseteq \exists r.C, C \sqsubseteq \bot}{D \sqsubseteq \bot}$$

In what follows, we call the set containing the above rule and the $\mathcal{ELH}_{\mathcal{R}+}$ rules in Sec. 2.2 the **R** rule set, which is sound and complete for $\mathcal{ELH}_{\bot,\mathcal{R}+}$ classification:

Lemma 1. *Let S be any set of TBox axioms closed under the **R** rule set, then $C \sqsubseteq D$ iff $C \sqsubseteq \bot \in S$ or $C \sqsubseteq D \in S$ for any C and D such that $C \sqsubseteq C \in S$, D occurs in $\mathcal{O}$ and $\bot \sqsubseteq \bot \in S$ if $\bot$ occurs in $\mathcal{O}$.*

With the **R** rules we can perform TBox reasoning:

Definition 1. ***(TBox Completion Closure)*** *Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ELH}_{\bot,\mathcal{R}+}$ ontology, its TBox completion closure, denoted by $S_{\mathcal{T}}$, is the smallest set of axioms closed under the rule set **R** such that:*

1. *for all $A \in \mathcal{CN}_{\mathcal{O}}$, $A \sqsubseteq A \in S_{\mathcal{T}}$;*
2. *$\bot \sqsubseteq \bot \in S_{\mathcal{T}}$ if $\bot$ occurs in $\mathcal{O}$.*

According to Lemma 1, we have $A \sqsubseteq C \in S_{\mathcal{T}}$ or $A \sqsubseteq \bot \in S_{\mathcal{T}}$ for any A and C where A is an atomic concept and C occurs in $\mathcal{T}$. This realises TBox classification.

4.2 ABox Completion Rules

Now we present the ABox completion rules for $\mathcal{ELH}_{\bot,\mathcal{R}+}$. Although $\mathcal{ELH}_{\bot,\mathcal{R}+}$ does not support nominals ($\{a\}$), we still denote individuals with nominals since this helps simplify the presentation: (i) ABox rules are more readable, as they have similar syntactic forms to the TBox ones, and (ii) some of the ABox rules can be unified. More precisely, we establish the following mappings as syntactic sugar:

$$\begin{aligned} C(a) &\Leftrightarrow \{a\} \sqsubseteq C \\ a \doteq b &\Leftrightarrow \{a\} \equiv \{b\} \\ a \dot{\neq} b &\Leftrightarrow \{a\} \sqcap \{b\} \sqsubseteq \bot \\ r(a,b) &\Leftrightarrow \{a\} \sqsubseteq \exists r.\{b\} \end{aligned}$$

Obviously, these mappings are semantically equivalent. In the rest of the paper, without further explanation, we treat the LHS and RHS of each of the above mappings as a *syntactic* variation of one another. Together with the mapping, all ABox axioms in the original $\mathcal{O}$ can be represented in a similar form of TBox axioms. Note that, axioms such as $C \sqsubseteq \{b\}$ and $C \sqsubseteq \exists r.\{b\}$ will not be in the ontology since they are invalid ABox axioms in $\mathcal{ELH}_{\bot,\mathcal{R}+}$.

We present the ABox completion rules as follows — we call them the **AR** rules, which should be applied *after* a complete closure $\mathbf{S}_\mathcal{T}$ is constructed from the **R** rules. In contrast to the **R** rules, the **AR** rules contain concepts $D_{(i)}$ and E that can take multiple forms including nominals. Thus the mapping between ABox and TBox axioms allows us to describe the rules in a more compact manner which would otherwise require additional rules to achieve the same purpose.

$$\mathbf{AR}_{\sqsubseteq}\frac{\{a\} \sqsubseteq D}{\{a\} \sqsubseteq E} : D \sqsubseteq E \in S_\mathcal{T} \cup \mathcal{A}$$

$$\mathbf{AR}^*_{\mathcal{H}}\frac{\{a\} \sqsubseteq \exists r.D}{\{a\} \sqsubseteq E} : \exists s.D \rightarrow E \in S_\mathcal{T}, r \sqsubseteq^*_\mathcal{O} s$$

$$\mathbf{AR}^*_{\mathcal{T}}\frac{\{a\} \sqsubseteq \exists r.D}{\exists t.\{a\} \rightarrow E} : \exists s.D \rightarrow E \in S_\mathcal{T},\ r \sqsubseteq^*_\mathcal{O} t \sqsubseteq^*_\mathcal{O} s,\ Trans(t) \in \mathcal{O}$$

$$\mathbf{AR}^-_{\sqcap}\frac{\{a\} \sqsubseteq D_1 \sqcap D_2}{\{a\} \sqsubseteq D_1; \{a\} \sqsubseteq D_2}$$

$$\mathbf{AR}^+_{\sqcap}\frac{\{a\} \sqsubseteq D_1, \{a\} \sqsubseteq D_2}{\{a\} \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \text{ occurs in } \mathcal{O}$$

$$\mathbf{AR}^+_{\exists}\frac{\{a\} \sqsubseteq D}{\exists s.\{a\} \rightarrow \exists s.D} : r \sqsubseteq^*_\mathcal{O} s, \exists r.D \text{ occurs in } \mathcal{O}$$

$$\mathbf{AR}_{\bot}\frac{\{b\} \sqsubseteq \exists r.\{a\}, \{a\} \sqsubseteq \bot}{\{b\} \sqsubseteq \bot}$$

$$\mathbf{AR}^*_{\bot}\frac{\{a\} \sqsubseteq \exists r.D}{\{a\} \sqsubseteq \bot} : D \sqsubseteq \bot \in S_\mathcal{T}$$

$$\mathbf{AR}_{\mathcal{H}}\frac{\{b\} \sqsubseteq \exists r.\{a\},\ \exists s.\{a\} \rightarrow E}{\{b\} \sqsubseteq E} : r \sqsubseteq^*_\mathcal{O} s$$

$$\mathbf{AR}_{\mathcal{T}}\frac{\{b\} \sqsubseteq \exists r.\{a\},\ \exists s.\{a\} \rightarrow E}{\exists t.\{b\} \rightarrow E} : r \sqsubseteq^*_\mathcal{O} t \sqsubseteq^*_\mathcal{O} s,\ Trans(t) \in \mathcal{O}$$

$$\mathbf{AR}^R_{\mathcal{H}}\frac{\{b\} \sqsubseteq \exists r.\{a\}}{\{b\} \sqsubseteq \exists s.\{c\}} : r \sqsubseteq^*_\mathcal{O} s,\ a = c \text{ or } \{a\} \sqsubseteq \{c\} \in \mathcal{A}$$

The **AR** rules deserve some explanations:

– There are clear correspondences between the **R** rules and **AR** rules. For example, $\mathbf{AR}_{\sqsubseteq}$ is an ABox counterpart of $\mathbf{R}_{\sqsubseteq}$ except that the context is explicitly a nominal, and TBox results are used as side conditions. The last rule $\mathbf{R}^R_{\mathcal{H}}$ is an additional rule to handle relations.

– Note that directly applying the **R** rules together with the **AR** rules could introduce unnecessary performance overheads such as axiom scheduling, processing and maintenance as we discussed in Sec. 3. In our approach, we separate TBox reasoning from ABox reasoning, and use TBox reasoning results as side conditions in ABox rules. This helps reduce memory usage and computation time.

Now we show below with an example on how the two-stage ABox reasoning works in operation. Suppose we have the following ontology:

$$PlanarStructure \sqsubseteq PhysicalStructure \tag{1}$$
$$PhysicalStructure \sqsubseteq GeneralisedStructure \sqcap \exists hasCountability.discrete \tag{2}$$
$$PlanarStructure \equiv \exists hasShape.(\exists hasAS.Laminar \sqcap Shape) \tag{3}$$
$$\{a\} \sqsubseteq \exists hasShape.\{b\} \tag{4}$$
$$\{b\} \sqsubseteq \exists hasAS.\{c\} \tag{5}$$
$$\{c\} \sqsubseteq Laminar \tag{6}$$
$$\{b\} \sqsubseteq Shape \tag{7}$$

We can see that (1)-(3) are TBox axioms and (4)-(7) are ABox axioms. Note that the input contains the assertions $hasShape(a, b)$, $hasAS(b, c)$, $Laminar(c)$ and $Shape(b)$, corresponding to (4)-(7) respectively. This conversion is expected to be performed before execution of the completion rules.

In the first stage, we compute the saturation of the TBox axioms (i.e., (1)-(3)) by applying the $\mathbf{R}$ rules. As an example, we illustrate how the axiom below is derived:

$$\exists hasShape.(\exists hasAS.Laminar \sqcap Shape) \sqsubseteq GeneralisedStructure \tag{8}$$

To begin, we apply $\mathbf{R}_{\sqsubseteq}$ on (1) and (3) to get (9), then again on (2) and (9) to get (10):

$$\exists hasShape.(\exists hasAS.Laminar \sqcap Shape) \sqsubseteq PhysicalStructure \tag{9}$$
$$\exists hasShape.(\exists hasAS.Laminar \sqcap Shape) \sqsubseteq GeneralisedStructure \sqcap \exists hasCountability.discrete \tag{10}$$

Lastly, we apply the $\mathbf{R}_{\sqcap}^{-}$-rule to (10) to get (8). Similarly, we infer all other TBox axioms by applying the completion rules repeatedly. Once saturation of the TBox rules is completed and the closure $\mathbf{S}_{\mathcal{T}}$ is constructed, we use the output $\mathbf{S}_{\mathcal{T}}$ from the first stage as part of input to the second stage to compute the saturation of the ABox axioms. Below, we demonstate how the ABox axiom $\{a\} \sqsubseteq GeneralisedStructure$ is inferred through the ABox rules. We start by applying $\mathbf{AR}_{\exists}^{+}$ on (6) and we get:

$$\exists hasAS.\{c\} \rightarrow \exists hasAS.Laminar \tag{11}$$

From (5) and (11) we apply the $\mathbf{AR}_{\mathcal{H}}$-rule to infer:

$$\{b\} \sqsubseteq \exists hasAS.Laminar \tag{12}$$

We then apply $\mathbf{AR}_{\sqcap}^{+}$ on (12) and (7) to obtain the following:

$$\{b\} \sqsubseteq \exists hasAS.Laminar \sqcap Shape \tag{13}$$

Similarly, we apply $\mathbf{AR}^{+}_{\exists}$ on (13), followed by $\mathbf{AR}_{\mathcal{H}}$ on (4)-(14):

$$\exists hasShape.\{b\} \rightarrow \exists hasShape.(\exists hasAS.Laminar \sqcap Shape) \quad (14)$$
$$\{a\} \sqsubseteq \exists hasShape.(\exists hasAS.Laminar \sqcap Shape) \quad (15)$$

Finally, we use the $\mathbf{AR}_{\sqsubseteq}$-rule on (15) and (10) to get $\{a\} \sqsubseteq GeneralisedStructure$. It can be converted back into assertion form $GeneralisedStructure(a)$.

Definition 2. ***(Ontology Completion Closure)*** *Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ELH}_{\perp,\mathcal{R}+}$ ontology, its ontology completion closure, denoted by* $\mathbf{S}$*, is the smallest set of axioms closed under the* ***AR*** *rule set such that:*

1. $S_{\mathcal{T}} \subseteq \mathbf{S}$;
2. $\mathcal{A} \subseteq \mathbf{S}$ *(axioms mapped as elaborated at the beginning of this section);*
3. *for all $a \in \mathcal{IN}_{\mathcal{O}}$, $\{a\} \sqsubseteq \{a\} \in X_{\mathcal{O}}$, and $\{a\} \sqsubseteq \top$ if $\top$ occurs in $\mathcal{O}$;*

Similar to the **R** rules, the above rules are also complete, sound and tractable. The soundness and tractability of rules are quite obvious. The completeness on ABox classification can be shown by the following Theorem:

Theorem 1. *For any $\mathcal{ELH}_{\perp,\mathcal{R}+}$ ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, we have either there is some $\{x\} \sqsubseteq \perp \in \mathbf{S}$, or*

1. *$\mathcal{O} \models D(a)$ only if $\{a\} \sqsubseteq D \in \mathbf{S}$ for D occurs in $\mathcal{O}$;*
2. *$\mathcal{O} \models r(a, b)$ only if $\{a\} \sqsubseteq \exists r.\{b\} \in \mathbf{S}$ for $r \in \mathcal{RN}_{\mathcal{O}}$.*

As we can see, the **AR** rules also preserve the feature that all premises of each rule have a same common part as context. Therefore, they still enjoy the lock-free feature in reasoning. In later sections, we will further elaborate this point.

4.3 Parallel Algorithms

In this section, we present the parallel algorithms corresponding to the $\mathcal{ELH}_{\perp,\mathcal{R}+}$ completion rules. We reuse some notions such as *context*, *activeContexts* queue, *scheduled* queue and *processed* set from the original TBox algorithm for $\mathcal{ELH}_{\mathcal{R}+}$ presented in [5] to realise the lock-free property. Most importantly, we need to make refinements to tailor the algorithm for $\mathcal{ELH}_{\perp,\mathcal{R}+}$ ABox reasoning. Here is a summary on how to deal with the challenges mentioned in Sec. 3 :

1. In our algorithm, reasoning is separated into two stages: the first stage is saturate(TBoxInput), where contexts are $\mathcal{ELH}_{\perp,\mathcal{R}+}$ concepts and the **R** rules are applied. The second is saturate(ABoxInput), where contexts are the (mapped) nominals and the **AR** rules are applied. See Algorithm 1 for details of the saturate() method.
2. Different from the TBox saturate algorithm in [5], in our algorithm, we parallelise the initialisation (line 3-8 of Algorithm 1) to improve efficiency. As mentioned in Sec. 3, initialisation could become non-trivial due to the large number of individuals. The introduction of parallelisation could help speed up these parts.

The revised saturation algorithm (Algorithm 1) is presented as follows. The saturation of an ontology is realised by first performing saturation of the TBox, the output of which (i.e., $S_{\mathcal{T}}$) is then used in the saturation of the ABox. The saturation of the ABox yields $\mathbf{S}$ which satisfies Theorem 1. All necessary tautology axioms must be added to the input prior to saturation. For TBox, we add axioms of the form $C \sqsubseteq C$ into $\mathbf{S}_{\mathcal{T}}$ for all concepts C such that $C \in \mathcal{CN}_{\mathcal{O}} \cup \{\bot\}$. Similarly, for ABox we add $\{a\} \sqsubseteq \{a\}$ into $\mathbf{S}$ for all individuals $a \in \mathcal{IN}_{\mathcal{O}}$.

Algorithm 1: saturate(input): saturation of axioms under inference rules

```
   Input: input (the set of input axioms)
   Result: the saturation of input is computed in context.processed
 1 activeContexts $\leftarrow \emptyset$;
 2 axiomQueue.addAll(input);
 3 loop
 4     axiom $\leftarrow$ axiomQueue.pop();
 5     if axiom $=$ null then break;
 6     for context $\in$ getContexts(axiom) do
 7         context.scheduled.add(axiom);
 8         activeContexts.activate(context);
 9 loop
10     context $\leftarrow$ activeContexts.pop();
11     if context $=$ null then break;
12     loop
13         axiom $\leftarrow$ context.scheduled.pop();
14         if axiom $=$ null then break;
15         process (axiom);
16     context.isActive $\leftarrow$ false;
17     if context.scheduled $\neq \emptyset$ then activeContexts.activate(context);
```

In the saturation (Algorithm 1), the $activeContexts$ queue is initialised with an empty set (line 1), and then all input axioms are added into an *axiomQueue* (line 2). After that, two main loops (lines 3-8 and lines 9-17) are sequentially parallelised. In the first main loop, multiple workers independently retrieve axioms from the *axiomQueue* (line 4), then get the contexts of the axioms (line 6), add the axioms into corresponding *scheduled* queues (line 7) and activate the contexts.

In the first loop of Algorithm 1 we need to get contexts of a given axiom (line 6), by calling the getContexts() method (Algorithm 2). As explained earlier, for TBox and ABox reasoning, the contexts are different. In ABox reasoning, only “nominals” can be contexts. Note that the getContexts() method is only used in Algorithm 1 during initialisation. The process() method (line 15 in Algorithm 1, to be discussed in Sec. 4.4), does not call the getContexts() method but directly get the contexts based on the form of input axiom. This is also different from the parallel TBox algorithm for $\mathcal{ELH}_{\mathcal{R}+}$ presented in [5].

Algorithm 2: getContext(axiom)

```
Input: an axiom
Result: the set of contexts that needs to be activated for the input axiom
1 result ← $\emptyset$;
2 if axiom contains no nominal then                // contexts for R rules
3     if axiom match $C \sqsubseteq D$ then result.add($C$);
4     if axiom match $D \sqsubseteq \exists r.C$ then result.add($C$);
5     if axiom match $\exists s.C \rightarrow E$ then result.add($C$);
6 else                                             // contexts for AR rules
7     if axiom match $\{a\} \sqsubseteq C$ then result.add ({a});
8     if axiom match $C \sqsubseteq \exists r.\{a\}$ then result.add ({a});
9 return result;
```

The activation of a context (Algorithm 3) is the same as in the TBox algorithm for $\mathcal{ELH}_{\mathcal{R}+}$ [5]: an atomic boolean value $isActive$ is associated with each context to indicate whether the context is already active. A context is added into the $activeContexts$ queue only if this value is $false$, which will be changed to $true$ at the time of activation. This procedure continues until the *axiomQueue* is empty.

Algorithm 3: activeContexts.activate(context)

```
Input: the context to be activated
1 if context.isActive.compareAndSwap(false, true) then
2     activeContexts.put(context);
```

In the second main loop of Algorithm 1, multiple workers independently retrieve contexts from the $activeContexts$ queue (line 10) and process its *scheduled* axioms (line 15). Once $context.scheduled$ is empty, $context.isActive$ is set to $false$ (line 16). A re-activation checking is performed (line 17) in case other workers have added new axioms into $context.scheduled$ while the last axiom is being processed (between line 14 and line 16). This procedure will continue until the $activeContexts$ queue is empty.

4.4 Cascading Processing

In this subsection, we describe the details of the $process()$ method, which covers items 2.(a), 2.(b), 2.(c) at the end of Sec. 2.2. As mentioned in Sec. 3, it is important to optimise the seemingly trivial parts, which could become non-trivial due to the large number of individuals. To address many of the issues mentioned in Sec. 3, we present a cascading processing procedure (Algorithm 4).

Algorithm 4: process(axiom) for context $\{a\}$

```
   Input: the axiom to be processed
1  if axiom match $\{a\} \sqsubseteq D$ then
2      if $D \in \{a\}$.subsumptions then break;
3      {a}.subsumptions.add (D);
       // For rule $\mathbf{AR}_{\sqsubseteq}$
4      for $E \in (D$.subsumptions $\cup$ $D$.originalTypes$)$ do
5          if $E \notin \{a\}$.subsumptions then
6              $\{a\}$.scheduled.add($\{a\} \sqsubseteq E$);
7              if $E$ match $\exists r.\{b\}$ then
8                  $\{b\}$.scheduled.add($\{a\} \sqsubseteq E$);
9                  activeContexts.activate($\{b\}$);
       // similarly for rules $\mathbf{AR}_H^*$, $\mathbf{AR}_T^*$, $\mathbf{AR}_\sqcap^-$, $\mathbf{AR}_\sqcap^+$, $\mathbf{AR}_\exists^+$,
          $\mathbf{AR}_\perp^*$ and $\mathbf{AR}_\perp$ right premise
10 if axiom match $\{b\} \sqsubseteq \exists r.\{a\}$ then
11     if $\langle r, \{b\}\rangle \in \{a\}$.predecessors then break;
12     $\{a\}$.predecessors.add($\langle r, \{b\}\rangle$);
13     if $\perp \in \{a\}$.subsumptions $\setminus$ $\{b\}$.subsumptions then
14         $\{b\}$.scheduled.add($\{b\} \sqsubseteq \perp$);
15         activeContexts.activate($\{b\}$);
       // similarly for rules $\mathbf{AR}_{\mathcal{H}}$, left premise, $\mathbf{AR}_T$, left
          premise and $\mathbf{AR}_{\mathcal{H}}^R$
16 if axiom match $\exists s.\{a\} \rightarrow E$ then
17     if $\langle s, E\rangle \in \{a\}$.implications then break;
18     $\{a\}$.implications.add($\langle s, E\rangle$);
19     for $r \in (\{a\}$.predecessors.keySet() $\cap$ $s$.subRoles$)$ do
20         for $\{b\} \in \{a\}$.predecessors.get($r$) do
21             if $E \notin \{b\}$.subsumptions then
22                 {b}.scheduled.add ({b}$\sqsubseteq$ E);
                   // similar as line 7-9
       // similarly for rules $\mathbf{AR}_T$, right premise
23 return result;
```

We match the form of input axiom once (line 1) and check whether it has been processed before (line 2); if not it will be added into the *processed* set (line 3). Based on the form of axiom, applicable completion rules can be determined. Meanwhile, checking if the conclusion is already in corresponding context's processed set can be performed (line 5). Once a completion rule has been applied, the conclusion axioms and their forms are determined. Once a conclusion is derived, its contexts and whether they are definitely the same as the current context are determined. The conclusion axioms can directly be added into corresponding *scheduled* queues (line 6 and 8). For the brevity of the paper, we only present the processing of some ABox axioms. Processing of TBox axioms and the other forms of ABox axioms can be done in a similar manner.

In Algorithm 4 certain axioms are maintained by several indexes to facilitate more efficient access. Most of them are the same as in the parallel TBox algorithm for $\mathcal{ELH}_{\mathcal{R}+}$ [5]. The additional one is $D.originalTypes$, which is used to maintain original ABox axioms:

$$\begin{aligned} D.originalTypes &= \{E | D \sqsubseteq E \in \mathcal{A}\}, \\ r.subRoles &= \{s | s \sqsubseteq^*_{\mathcal{O}} r\}, \\ C.subsumptions &= \{D | C \sqsubseteq D \in \text{processed}\}, \\ C.predecessors &= \{\langle r, D\rangle | D \sqsubseteq \exists r.C \in \text{processed}\}, \\ C.implications &= \{\langle r, E\rangle | \exists r.C \rightarrow E \in \text{processed}\}, \end{aligned}$$

5 Evaluation

We implemented our algorithms in our PEL reasoner (written in JAVA). Inspired by the ELK reasoner [5] , we also use thread-safe datatypes `ConcurrentLinkedQueue` for all the queues, including $activeContexts$, $axiomQueue$ and $scheduled$. And we use `AtomicBoolean` for the $isActive$ value of a context thus its $compareAndSwap$ operation is atomic and thread-safe. The indexes we used in Algorithm 4 are implemented with normal `HashSet` and `HashMap`. We use OWL API to parse ontologies.

To compare our system against sequential reasoners we use the Amazon Elastic Computer Cloud (EC2) High-CPU Extra Large Instance [7]. It has 7 GB of memory and 8 cores with 2.5 EC2 compute units each, where each EC2 unit " provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor". The OS is 64-bit Linux platform and running JVM version 1.6.0_20 with 7 GB memory. We run our implementation with sequential reasoners Pellet 2.2.2, FaCT++ 1.5.2, HermiT 1.3.2 and (the OWL 2 EL reasoner in) TrOWL 0.8 because they (except HermiT) implemented the EL algorithm and support ABox reasoning. Other reasoners, including dedicated EL reasoners such as the OWL API-compliant CEL, jCEL and Snorocket, etc. and consequence-driven reasoner CB and ELK, do not fully support ABox reasoning yet.

Our test cases include a slightly simplified real world ontology VICODI, and a real world TBox NotGalen with generated ABox [8]. The VICODI [9] ontology is developed to represent the history of Europe. It has a simple TBox and moderate number of individuals. NotGalen$^-$ is extracted from an earlier version of Galen [10] by removing functional role assertions. It contains a moderate-size TBox and no ABox. To populate the ontology we use the SyGENiA system [14] to generate ABoxes for a small part of the Galen ontology and we combined the generated ABoxes of different sizes with the NotGalen$^-$ TBox and aligned the namespaces to make reasoning more complicated; in this way, we have test ontologies NG-1, NG-2, NG-5 etc. Such ABoxes are not completely random because, as generated by SyGENiA, they cover axioms that can lead to

[7] http://aws.amazon.com/ec2/instance-types/

[8] Our test ontologies can be found at http://www.box.net/shared/qok98u39s9lrmy3ie2n1.

[9] http://www.vicodi.org/about.htm

[10] http://www.opengalen.org/

all possible sources of incompleteness w.r.t. a TBox and certain query. Being able to handle such ABoxes means that the reasoner won't miss any result when dealing with any real-world ABoxes. The stats of our ontologies are illustrated in the Table 2.

For each ontology, we perform ABox classification, i.e. to compute the atomic types of all the individuals and the atomic relations between all pairs of individuals. If such assertion are not "pre-computable" when a reasoner classifies the ontology, we use the reasoner API functions to retrieve these results to make sure they are computed. The time shown in our evaluation is the overall computation time. Results of sequential reasoners reasoners are presented in Table 2. Results of our implementation PEL are presented in Table 3. The timeout is one hour. Time unit is second.

Table 2. Ontologies and Results of Sequential Reasoners (in sec)

Ontology	$\|\mathcal{CN}\|$	$\|\mathcal{RN}\|$	$\|\mathcal{IN}\|$	$\|\mathcal{A}\|$	TrOWL	Pellet	HermiT	FaCT++
VICODI	184	10	29614	114164	2.014	9.971	13.138	timeout
NG-1	2748	413	4236	8008	4.284	210.945	307.93	timeout
NG-2			12161	23976	9.342	757.379	timeout	timeout
NG-5			47756	118458	28.947	timeout	timeout	timeout
NG-8			78899	278365	63.833	timeout	timeout	timeout
NG-13			97995	665304	143.288	timeout	timeout	timeout

Table 3. Results of PEL (in sec)

Ontology	1 worker	2 workers	4 workers	6 workers
VICODI	1.136	1.05	1.054	1.059
NG-1	2.339	1.361	1.169	1.069
NG-2	3.025	2.939	2.848	2.77
NG-5	6.427	6.004	5.114	5.125
NG-8	12.604	10.474	9.449	9.75
NG-13	23.609	20.853	16.877	17.539

From the comparison between Table 2 and 3 we can see that PEL is in general faster than sequential reasoners, especially when more workers are used. PEL is also good when dealing with combination of complex TBox and large ABox. For example, in the relatively simpler VICODI ontology, PEL is about 2 times faster than TrOWL, which is also highly optimised for $\mathcal{EL}$ reasoning. While in the more complex NotGalen$^-$ ontologies with a large ABox, PEL is up to 6 times faster than TrOWL with one worker, and up to about 8-9 times faster with multiple workers.

To further evaluate the scalability of PEL, we generated a different set of NotGalen$^-$ ontologies with larger numbers of individuals, denoted by NGS-1, NGS-5, NGS-10, etc. And we use PEL to reason with these ontologies on a EC2 High-Memory Quadruple Extra Large Instance which has 8 virtual cores with 3.25 EC2 units each and 60 G memory allocated to JVM. Results are shown in Table 4.

From the comparison between different numbers of workers in Table 3 and Table 4 we can see that multiple parallel workers can indeed improve the reasoning performance, even when the ontology contains complex TBox and very large number of indi-

Table 4. Results of PEL (in sec) for Scalability Tests

Ontology	$\lvert\mathcal{IN}\rvert$	$\lvert\mathcal{A}\rvert$	1 worker	2 workers	4 workers	6 workers
NGS-1	4031	8001	1.396	0.977	0.757	0.676
NGS-5	62572	119832	3.81	2.885	2.376	2.341
NGS-10	211408	437637	10.282	8.007	7.04	6.493
NGS-20	596007	1642876	52.753	40.208	40.228	34.507
NGS-30	866136	3542257	108.252	90.72	81.877	77.453
NGS-40	971222	6036910	172.743	146.411	131.536	129.827
NGS-50	995985	9025426	270.806	234.905	189.706	190.303

viduals. In general, the improvement is most profound from 1 worker to 2 workers, and start to decrease when more workers are involved. With more than 4 workers, the performance may even decrease. We believe one of the potential reasons is that although the CPU cores can work in parallel, the RAM bandwidth is limited and RAM access is still sequential. In relatively “light-weight” ABox reasoning with large ABox, the RAM access will be enormous and very often so that multiple workers will have to compete for RAM access. This makes memory I/O a potential bottleneck of parallelisation and wastes CPU cycles. In our algorithm, especially the cascading processing, we have already tried to reduce unnecessary memory I/O. A better management of memory will be an important direction of our future work.

6 Conclusion

In this paper we extended early related work to present a parallel ABox reasoning approach to $\mathcal{ELH}_{\bot,\mathcal{R}+}$ ontologies. We have proposed new completion rules and show that they are complete and sound for ABox reasoning. We have revised the lock-free saturation procedure with optimisations that take the features of ABox reasoning into account. Particularly, we separate TBox and ABox reasoning to simplify derivation and parallise many seemingly trivial steps to improve efficiency and reduce memory access. Our evaluation shows that ABox reasoning can benefit from parallisation. Even with our naive implementation, we can outperform highly optimised $\mathcal{EL}$ reasoners.

The evaluation results suggested that improving performance with more than 4 workers becomes difficult, which is also observed in [5]. In our future work we will further investigate its reason and pay special attention on the management of memory.

Acknowledgement

This work is partially funded by the K-Drive and ITA projects. We would also like to thank reviewers for their very constructive and helpful comments.

References

1. Mina Aslani and Volker Haarslev. Parallel tbox classification in description logics –first experimental results. In *Proceeding of the 2010 conference on ECAI 2010: 19th European*

Conference on Artificial Intelligence, pages 485–490, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

2. Alex Borgida and Luciano Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, 1, 2003.
3. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.
4. Aidan Hogan, Jeff Z. Pan, Axel Polleres, and Stefan Decker. Saor: Template rule optimisations for distributed reasoning over 1 billion linked data triples. In *International Semantic Web Conference (1)*, pages 337–353, 2010.
5. Yevgeny Kazakov, Markus Krötzsch, and František Simančik. Concurrent classification of el ontologies. In *ISWC*, 2011. To appear.
6. Thorsten Liebig and Felix Müller. Parallelizing tableaux-based description logic reasoning. In *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems - Volume Part II*, OTM'07, pages 1135–1144, Berlin, Heidelberg, 2007. Springer-Verlag.
7. Raghava Mutharaju Frederick Maier, , and Pascal Hitzler. A mapreduce algorithm for el+. In *Proc. of International Worshop of Description Logic (DL2010)*, 2010.
8. Adam Meissner. Experimental analysis of some computation rules in a simple parallel reasoning system for the $\mathcal{ALC}$ description logic. *Applied Mathematics and Computer Science*, 21(1):83–95, 2011.
9. Eyal Oren, Spyros Kotoulas, George Anadiotis, Ronny Siebes, Annette ten Teije, and Frank van Harmelen. Marvin: Distributed reasoning over large-scale semantic web data. *Web Semant.*, 7:305–316, December 2009.
10. Anne Schlicht and Heiner Stuckenschmidt. Distributed resolution for alc. In *Description Logics Workshop*, 2008.
11. Anne Schlicht and Heiner Stuckenschmidt. Distributed resolution for expressive ontology networks. In *Proceedings of the 3rd International Conference on Web Reasoning and Rule Systems*, RR '09, pages 87–101, Berlin, Heidelberg, 2009. Springer-Verlag.
12. Luciano Serafini and Andrei Tamilin. Drago: Distributed reasoning architecture for the semantic web. In *In ESWC*, pages 361–376. Springer, 2005.
13. Ramakrishna Soma and V. K. Prasanna. Parallel inferencing for owl knowledge bases. In *Proceedings of the 2008 37th International Conference on Parallel Processing*, ICPP '08, pages 75–82, Washington, DC, USA, 2008. IEEE Computer Society.
14. Giorgos Stoilos, Bernardo Cuenca Grau, and Ian Horrocks. How incomplete is your semantic web reasoner? In *Proc. of AAAI 10*, pages 1431–1436. AAAI Publications, 2010.
15. Herman J. ter Horst. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. *J. Web Sem.*, 3(2-3):79–115, 2005.
16. Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri Bal. Owl reasoning with webpie: calculating the closure of 100 billion triples. In *Proceedings of the Seventh European Semantic Web Conference*, LNCS. Springer, 2010.
17. Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank van Harmelen. Scalable distributed reasoning using mapreduce. In *Proceedings of the ISWC '09*, volume 5823 of *LNCS*. Springer, 2009.
18. Jesse Weaver and James A. Hendler. Parallel materialization of the finite rdfs closure for hundreds of millions of triples. In Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*, pages 682–697. Springer, 2009.
19. Gang Wu, Guilin Qi, and Jianfeng Du. Finding all justifications of owl entailments using tms and mapreducec. In *the ACM Conference on Information and Knowledge Management*, 2011.