Optimising Ontology Stream Reasoning with Truth Maintenance System [*]

Yuan Ren
Department of Computing Science
University of Aberdeen, Aberdeen
AB24 3UE, UK
y.ren@abdn.ac.uk

Jeff Z. Pan
Department of Computing Science
University of Aberdeen, Aberdeen
AB24 3UE, UK
jeff.z.pan@abdn.ac.uk

ABSTRACT

So far researchers in the Description Logics / Ontology communities mainly consider ontology reasoning services for static ontologies. The rapid development of the Semantic Web and its emerging data ask for reasoning technologies for dynamic knowledge streams. Existing work on stream reasoning is focused on lightweight languages such as RDF and RDFS. In this paper, we introduce the notion of Ontology Stream Management System (OSMS) and present a stream-reasoning approach based on Truth Maintenance System (TMS). We present optimised $\mathcal{EL}^{++}$ algorithm to reduce memory consumption. Our evaluations show that the optimisation improves TMS-enabled $\mathcal{EL}^{++}$ reasoning to deal with relatively large volumes of data and update efficiently.

Categories and Subject Descriptors

I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Inference engines*

General Terms

Algorithms

Keywords

Ontology, Stream Reasoning, Truth Maintenance System

1. INTRODUCTION

Ontologies are the knowledge infrastructures of the Semantic Web and many intelligent systems. The standard Web Ontology Language OWL is based on the Description Logics (DLs). So far researchers in the DL/ Ontology communities mainly consider ontology reasoning services for static ontologies. However, in real world data and knowledge is usually subject to change. Parsia et. al. [13] described several typical scenarios such as ontology editing, web service matchmaking, sensor networks and mobile semantic web [11]. Considering the following running example:

[*]This paper is an extension of [14], an earlier work presented in a workshop.

EXAMPLE 1. *Given the schedule of a conference and researchers' personal information and activities, such as who is attending which events, a knowledge management system would be able to collect data and provide recommendations, in real time.*

In this paper, we assume such data has been extracted and formalised, and focus on the data processing and reasoning aspect. Due to the real-time nature of these user generated information and ongoing events, it is adequate to consider them as data streams.

There have been many works regarding **querying streaming data** on the semantic web. A. Bolles et al. [7] proposed an extension of SPARQL to process data streams. Similarly, C-SPARQL [5], another extension of SPARQL can continuously query from a RDF knowledge base. More related work focus on **continuously and incrementally updating and materialising ontological knowledge bases**. R. Volz et al. [15] adopted the Delete and Re-derive (DRed) algorithm [9] from traditional data stream management systems and proposed a declarative variant of it. When change occurs, the "stream reasoner" first overestimates the consequences of the deletion, then "cash-back" the over-deleted consequences that can be derived by other facts, and finally adds new entailments that are derived from the new facts. Further optimisation was proposed [6] with an additional assumption that the time window of stream is fixed and known to the stream reasoner. The relation between a consequence and a time point can be maintained so that when the time comes, the stream reasoner expires corresponding consequences. In their evaluation, this approach outperforms a naive approach with up to 13% of RDF triples changed, which is better than the 3% limit of [15]. However, these approaches are still limited to relatively simple languages such as RDF, RDFS. The optimisation also requires a known fixed time window. Otherwise, the optimisation can not work. This also means deleting axioms in real-time is not allowed. For example, if data collected from different sources make the ontology inconsistent, such inconsistency can not be actively resolved by removing conflicting knowledge.

In this paper, we introduce the notion of ontology stream management system (OSMS). We present an ontology stream reasoning approach by applying Truth Maintenance Systems (TMS [8]) for OSMS. A TMS maintains not only reasoning results, but also intermediate results and the deduction relations among them. Thus on the one hand, impacted results of removed knowledge can be retrieved without prior knowledge of fixed time window. On the other hand, further inference can be carried on from the intermediate results, instead of the original asserted knowledge. Both will significantly increase the efficiency of stream reasoning to facilitate real-time response. However, TMS has a well-known disadvantage of excessive memory consumption. To address this issue, we present an optimised algorithm for $\mathcal{EL}^{++}$, the logic underpinning OWL 2 EL, by reducing the number of unnecessary intermediate results. Our evaluation shows that our approach can outperform naive re-computation with relatively large volumes of knowledge base (up to 30000 axioms) and updates (up to 10%).

2. BACKGROUND

A signature $\Sigma = (\mathcal{CN}, \mathcal{RN}, \mathcal{IN})$ consists of three disjoint sets $\mathcal{CN}$, $\mathcal{RN}$, $\mathcal{IN}$, where $\mathcal{CN}$ is the set of atomic concepts, $\mathcal{RN}$ is the set of atomic roles and $\mathcal{IN}$ is the set of individuals. Given a signature, $\top$ the top concept, $\bot$ the bottom concept, A an atomic concept, a an individual, r an atomic role, $\mathcal{EL}^{++}$ concept expressions C, D can be composed with the following constructs:

$$\top \mid \bot \mid A \mid C \sqcap D \mid \exists r.C \mid \{a\}$$

We slightly abuse the notion of atomic concepts to include also $\top$, $\bot$ and nominals (i.e. concepts of form $\{a\}$). Given a knowledge base $\mathbf{K}$, we use $CN_{\mathbf{K}}$, $RN_{\mathbf{K}}$, $IN_{\mathbf{K}}$ to denote the set of atomic concepts, atomic roles and individuals in $\mathbf{K}$, respectively.

A DL ontology $\mathcal{O} =< \mathcal{T}, \mathcal{A} >$ is composed of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. A TBox is a set of concept and role axioms. $\mathcal{EL}^{++}$ supports general concept inclusion axioms (GCIs, e.g. $C \sqsubseteq D$) and role inclusion axioms (RIs, e.g. $r \sqsubseteq s$, $r_1 \circ \cdots \circ r_n \sqsubseteq s$). If $C \sqsubseteq D$ and $D \sqsubseteq C$, we write $C \equiv D$. An ABox is a set of concept assertion axioms, e.g. $a : C$, role assertion axioms, e.g. $(a, b) : R$, and individual equality axioms, e.g. $a = b$, and individual inequality axioms, e.g. $a \neq b$.

EXAMPLE 2. *Here are some simple $\mathcal{EL}^{++}$ ontologies:*
Ontology $\mathcal{O}$: { $ActiveTalk \sqsubseteq \exists in.Session$, $hasTopic \circ interest \sqsubseteq recommend$, $\exists recommend.\{David\} \sqsubseteq Talk4Dave$, $ActiveTalk \sqcap Talk4Dave \sqsubseteq TargetTalk$, $(ontology, Daivd) : interest$, $(talk1, ontology) : hasTopic$, $(talk2, ontology) : hasTopic$, } specifies that an ActiveTalk *must be presented in some* Session*; A talk can be recommended to someone if the talk has topic that interests the person; A talk is for David if it can be recommended to David and such a talk will be a $TargetTalk$ if it is active; It also says that David is interested in ontology. There are two talks $talk1$ and $talk2$ that have topic about ontology.*
$\mathcal{O}(0) = \{talk0 : ActiveTalk\}$ shows that $talk0$ is an $ActiveTalk$.
$\mathcal{O}(1) = \{talk1 : ActiveTalk\}$ shows that $talk1$ is an $ActiveTalk$.
$\mathcal{O}(2) = \{talk2 : ActiveTalk\}$ shows that $talk2$ is an $ActiveTalk$.

If an axiom α can be derived from an ontology $\mathcal{O}$, we say that α is entailed by $\mathcal{O}$, denoted by $\mathcal{O} \models \alpha$. α is an entailment of $\mathcal{O}$. For an ontology $\mathcal{O}$ and its entailment α, a set of axioms $J_{\mathcal{O}}(\alpha) \subseteq \mathcal{O}$ is a justification of α *iff* $J_{\mathcal{O}}(\alpha) \models \alpha$ and $J' \not\models \alpha$ for all $J' \subset J_{\mathcal{O}}(\alpha)$. For $\mathcal{EL}^{++}$, a single justification can be computed in PTIME [4].

Franz Baader et. al. [1] presents a completion-based algorithm to classify an $\mathcal{EL}^{++}$ TBox. Given an $\mathcal{EL}^{++}$ TBox $\mathcal{T}$, it can be *normalised* in linear time [3] into a **normal form** TBox $\mathcal{T}'$ in which all axioms are in one of the following forms, where $A_i \in \mathcal{CN}_{\mathcal{T}'}$ and $r_i \in \mathcal{RN}_{\mathcal{T}'}$, by introducing fresh atomic concepts and roles:

$$\begin{array}{rclrcl} A_1 & \sqsubseteq & A_2, & A_1 & \sqsubseteq & \exists r.A_2, \\ A_1 \sqcap \cdots \sqcap A_n & \sqsubseteq & A_{n+1}, & \exists r.A_1 & \sqsubseteq & A_2, \\ r_1 \circ r_2 & \sqsubseteq & r_3, & r_1 & \sqsubseteq & r_2. \end{array}$$

For any two $A, B \in \mathcal{CN}_{\mathcal{T}}$, $\mathcal{T} \models A \sqsubseteq B$ iff $\mathcal{T}' \models A \sqsubseteq B$ [2]. Given a normalised $\mathcal{EL}^{++}$ TBox $\mathcal{T}$, the algorithm uses a set of completion rules (Table 1, in **R6** $X \leadsto_R A$ iff there exists $C_1, \ldots, C_k \in \mathcal{CN}_{\mathcal{T}}$ s.t. $C_1 = X$ or $C_1 = \{b\}$, $C_j \sqsubseteq \exists r_j.C_{j+1}$ for some $r_j \in \mathcal{RN}_{\mathcal{T}}$ $(1 \leq j \leq k)$ and $C_k = A$) to compute, for each two $A, B \in \mathcal{CN}_{\mathcal{T}}$, entailed subsumption $\mathcal{T} \models A \sqsubseteq B$.

Reasoning with rules **R1-R8** is PTIME-Complete [2]. ABox reasoning can also be reduced to TBox reasoning with these rules [2].

Table 1: $\mathcal{EL}^{++}$ TBox completion rules (no datatypes)

R1	If $X \sqsubseteq A$, $A \sqsubseteq B$ then $X \sqsubseteq B$
R2	If $X \sqsubseteq A_1, \ldots, A_n$, $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B$ then $X \sqsubseteq B$
R3	If $X \sqsubseteq A$, $A \sqsubseteq \exists r.B$ then $X \sqsubseteq \exists r.B$
R4	If $X \sqsubseteq \exists r.A$, $A \sqsubseteq A'$, $\exists r.A' \sqsubseteq B$ then $X \sqsubseteq B$
R5	If $X \sqsubseteq \exists r.A$, $A \sqsubseteq \bot$ then $X \sqsubseteq \bot$
R6	If $X, A \sqsubseteq \{a\}$, $X \leadsto_R A$ then $X \sqsubseteq A$
R7	If $X \sqsubseteq \exists r.A$, $r \sqsubseteq s$ then $X \sqsubseteq \exists s.A$
R8	If $X \sqsubseteq \exists r_1.A$, $A \sqsubseteq \exists r_2.B$, $r_1 \circ r_2 \sqsubseteq r_3$, then $X \sqsubseteq \exists r_3.B$

Alternatively, we can internalize ABox axioms into TBox axioms as follows to perform reasoning. In the rest of the paper, we always internalize ABox into TBox.

$$\begin{array}{lclclcl} a : C & \leadsto & \{a\} \sqsubseteq C & \quad & (a,b) : r & \leadsto & \{a\} \sqsubseteq \exists r.\{b\} \\ a \doteq b & \leadsto & \{a\} \equiv \{b\} & & a \dot{\neq} b & \leadsto & \{a\} \sqcap \{b\} \sqsubseteq \bot \end{array}$$

3. ONTOLOGY STREAM MANAGEMENT SYSTEMS

In this section, we introduce the notion of ontology stream management systems (OSMS). A typical ontology stream management system is illustrated in Fig. 1. It can retrieve certain information

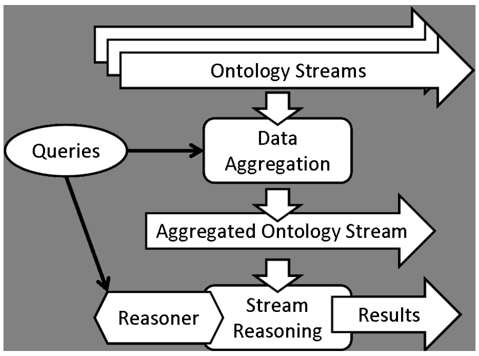


Figure 1: Ontology Stream Management System

from streams. **Data Aggregation** is the type of retrieval that does not require reasoning. For example, in the above stream, we can ask which takes are active. This type of retrieval has been widely addressed by, e.g., the C-SPARQL proposals. **Stream Reasoning** requires reasoning to be performed on the stream to yield correct answers. Their output are streams of results.

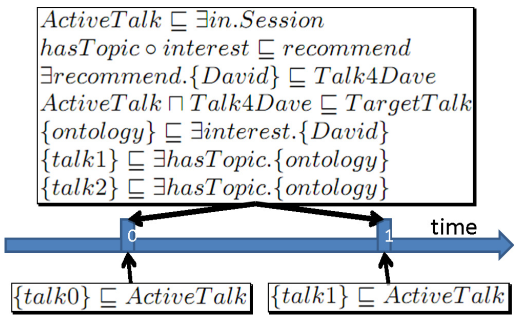
In stream reasoning [6] a streaming RDF knowledge base is defined as a set of pairs (α, t) where each RDF triple α is associated with a timestamp t. By grouping triples of the same timestamp t we have the ontology at time point t. Such a versioning idea resembles the concept of *Linear Version Space* (LVS) [10], which is a sequence of ontologies $S = (\mathcal{O}_0, \mathcal{O}_1, \ldots, \mathcal{O}_n)$. We recast these concepts and present a unified definition:

DEFINITION 1. ***(Discrete Ontology Stream)*** *An discrete ontology stream $\mathcal{O}^n_m$ from point of time m to point of time n is a sequence of ontologies $\mathcal{O}^n_m(t_0), \mathcal{O}^n_m(t_1), \ldots, \mathcal{O}^n_m(t_x)$ where $t_0 = m \leq t_1 \leq \cdots \leq t_x = n$. $\mathcal{O}^n_m(t_i)$ is called a* snapshot *at t_i.*

For conciseness, in the rest of the paper we use the name *ontology stream* or *stream* for short. We also assume that a stream always starts from time point 0 and is updated at each following integer time point, i.e. $\mathcal{O}^n_0 = \mathcal{O}^n_0(0), \mathcal{O}^n_0(1), \ldots \mathcal{O}^n_0(n)$. A change from

$\mathcal{O}_0^n(i)$ to $\mathcal{O}_0^n(i+1)$ is an update. When a "new" update occurs, the stream is consequently updated from $\mathcal{O}_0^n$ to $\mathcal{O}_0^{n+1}$. It's important to note that the updates are not necessarily known to stream reasoners in advance. That means any axiom can be added into, or removed from the changing ontology at any time.

From Example 2 we construct a stream $\mathcal{O}_0^1 = \mathcal{O} \cup \mathcal{O}(0), \mathcal{O} \cup \mathcal{O}(1)$. The stream is illustrated in Fig. 2. We assume all ABox axioms have been internalised. This stream specifies that the fea-

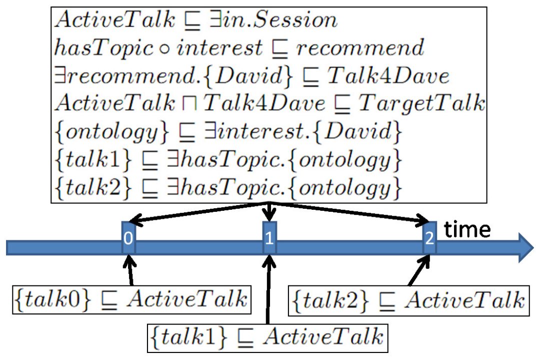


$ActiveTalk \sqsubseteq \exists in.Session$
$hasTopic \circ interest \sqsubseteq recommend$
$\exists recommend.\{David\} \sqsubseteq Talk4Dave$
$ActiveTalk \sqcap Talk4Dave \sqsubseteq TargetTalk$
$\{ontology\} \sqsubseteq \exists interest.\{David\}$
$\{talk1\} \sqsubseteq \exists hasTopic.\{ontology\}$
$\{talk2\} \sqsubseteq \exists hasTopic.\{ontology\}$
0
1
time
$\{talk0\} \sqsubseteq ActiveTalk$
$\{talk1\} \sqsubseteq ActiveTalk$

Figure 2: Example Stream

ture of $ActiveTalk$, the role chain of $recommend$, the condition of $Talk4Dave$ and $TargetTalk$, $David$'s interest on $ontology$, and the topics of the two talks preserve from time point 0 to 1. $talk0$ is active in time point 0 and $talk1$ is active in time point 1.

In this paper, we focus on such stream reasoning services. More formally, for a stream $\mathcal{O}_0^n$ and a reasoning problem Q, applications usually ask for reasoning results of Q on snapshot $\mathcal{O}_0^n(i)$, denoted by $Ans(\mathcal{O}_0^n(i), Q)$. For example, with the information in Fig. 2, we can ask for all ongoing talks that are interesting to David, which can be regarded as $Q =$*retrieving instances of* $TargetTalk$. Such a request can be simply fulfilled by applying the completion rules on the snapshots. By manual checking we know that $Ans(\mathcal{O}_0^1(0), Q) = \emptyset$ and $Ans(\mathcal{O}_0^1(1), Q) = \{talk1 : TargetTalk\}$. If we update $\mathcal{O}_0^1$ to $\mathcal{O}_0^2$ with $\mathcal{O}_0^2(2) = \mathcal{O} \cup \mathcal{O}(2)$. A new talk $talk2$ is being presented at time point 3 (Fig. 3). Now David wants to



$ActiveTalk \sqsubseteq \exists in.Session$
$hasTopic \circ interest \sqsubseteq recommend$
$\exists recommend.\{David\} \sqsubseteq Talk4Dave$
$ActiveTalk \sqcap Talk4Dave \sqsubseteq TargetTalk$
$\{ontology\} \sqsubseteq \exists interest.\{David\}$
$\{talk1\} \sqsubseteq \exists hasTopic.\{ontology\}$
$\{talk2\} \sqsubseteq \exists hasTopic.\{ontology\}$
0
1
2
time
$\{talk0\} \sqsubseteq ActiveTalk$
$\{talk2\} \sqsubseteq ActiveTalk$
$\{talk1\} \sqsubseteq ActiveTalk$

Figure 3: Updated Stream

know, what is $Ans(\mathcal{O}_0^2(0), Q)$, given the results of $Ans(\mathcal{O}_0^1(0), Q)$?

We can directly compute the new results on $\mathcal{O}_0^2(2)$. We call this approach the *naive approach*. This approach is not efficient because the time of computing the new results is determined by the entire ontology, instead of the changed part. Even if the change is very small, the updating of results can take very long time if the ontology is big. More interestingly, people would like to compute results on the latest snapshot based on results on the previous snapshot, so that partial results can be reused without re-computation.

As we mentioned in the last section, existing work is limited to relatively simple languages and have certain restrictions. In the next sections, we propose a truth maintenance system-based approach for $\mathcal{EL}^{++}$ and which requires no fixed time window.

4. STREAM REASONING WITH TMS

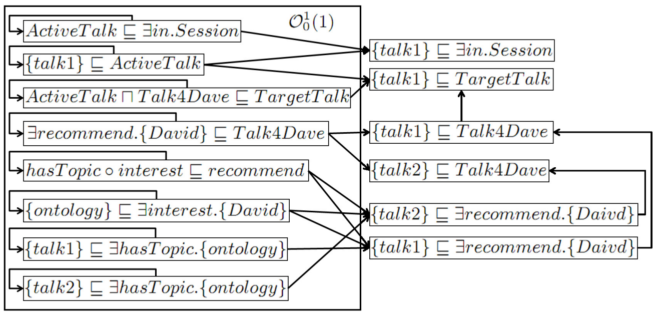
In this section, we present how to provide stream reasoning in OSMS. Different from the naive approach, which re-compute everything from scratch, we propose to maintain the (intermediate) results and their inferencing relations, so that when updating, entailments affected or not affected by the removed axioms can be easily distinguished and intermediate results can be re-used. Such maintenance can be realised by a Truth Maintenance System (TMS [8]).

A TMS maintains both beliefs and their dependencies in the form of a dependency network, in which nodes are beliefs and edges are the inference steps from which the nodes are derived.

DEFINITION 2. ***(Truth Maintenance System)*** *Given an ontology $\mathcal{O}$, a TMS $G_{\mathcal{O}} = \langle N_{\mathcal{O}}, E_{\mathcal{O}} \rangle$ of $\mathcal{O}$ is a directed graph such that (1) $\mathcal{O} \subseteq N_{\mathcal{O}}$; (2)$N_{\mathcal{O}} \subseteq \{\alpha | \mathcal{O} \models \alpha\}$; (3) for any $\alpha \in N_{\mathcal{O}}$, $\{\beta | (\beta, \alpha) \in E_{\mathcal{O}}\}$ is a minimal set of axioms that entails α.*

The 3rd property is important such that if all inbound nodes of an entailment are preserved, then the entailment is preserved. Otherwise the entailment should be removed unless a different set of axioms can entail it.

It is obvious that a node can have multiple inbound edges if the entailment is derived from multiple other entailments. Tautology axioms do not have any inbound edges. Asserted axioms only have inbound edges from themselves. It is also apparent that an ontology can have multiple or even infinite TMSes, depending on how many entailments are preserved. For example, applying **R1-8** in Table 1, a TMS of $\mathcal{O}_0^1(1)$ from our running example is illustrated in Fig. 4.

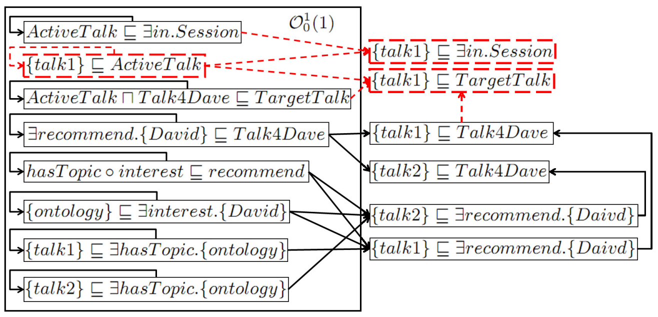


$\mathcal{O}_0^1(1)$
$ActiveTalk \sqsubseteq \exists in.Session$
$\{talk1\} \sqsubseteq ActiveTalk$
$ActiveTalk \sqcap Talk4Dave \sqsubseteq TargetTalk$
$\exists recommend.\{David\} \sqsubseteq Talk4Dave$
$hasTopic \circ interest \sqsubseteq recommend$
$\{ontology\} \sqsubseteq \exists interest.\{David\}$
$\{talk1\} \sqsubseteq \exists hasTopic.\{ontology\}$
$\{talk2\} \sqsubseteq \exists hasTopic.\{ontology\}$
$\{talk1\} \sqsubseteq \exists in.Session$
$\{talk1\} \sqsubseteq TargetTalk$
$\{talk1\} \sqsubseteq Talk4Dave$
$\{talk2\} \sqsubseteq Talk4Dave$
$\{talk2\} \sqsubseteq \exists recommend.\{Daivd\}$
$\{talk1\} \sqsubseteq \exists recommend.\{Daivd\}$

Figure 4: TMS of $\mathcal{O}_0^1$ (1) $G_{\mathcal{O}_0^1(1)}$

We first show how the TMS can be used in stream reasoning. After that we explain how the nodes and edges of a TMS can be constructed. Suppose we have an ontology stream $\mathcal{O}_0^n$, a reasoning request Q and a TMS $G_{\mathcal{O}_0^n(n)}$ containing all elements of $Ans(\mathcal{O}_0^n(n), Q)$, when an update occurs, i.e. we have a new snapshot $\mathcal{O}_0^{n+1}(n+1)$. We can update the entailments in $N_{\mathcal{O}_0^n(n)}$ by updating TMS $G_{\mathcal{O}_0^{n+1}(n+1)}$ out of the old one:

1. Initialise $G_{\mathcal{O}_0^{n+1}(n+1)} = G_{\mathcal{O}_0^n(n)}$;
2. Remove all the erased axioms and their reachable nodes from $N_{\mathcal{O}_0^{n+1}(n+1)}$. The erased TMS of our example is illustrated in Fig. 5, where the red node and edges are removed;
3. Add all new axioms in $\mathcal{O}_0^{n+1}(n+1)$ to $N_{\mathcal{O}_0^{n+1}(n+1)}$;

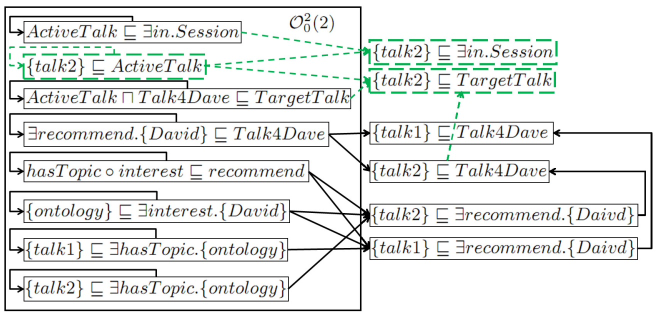


$\mathcal{O}_0^1(1)$
$ActiveTalk \sqsubseteq \exists in.Session$
$\{talk1\} \sqsubseteq ActiveTalk$
$ActiveTalk \sqcap Talk4Dave \sqsubseteq TargetTalk$
$\exists recommend.\{David\} \sqsubseteq Talk4Dave$
$hasTopic \circ interest \sqsubseteq recommend$
$\{ontology\} \sqsubseteq \exists interest.\{David\}$
$\{talk1\} \sqsubseteq \exists hasTopic.\{ontology\}$
$\{talk2\} \sqsubseteq \exists hasTopic.\{ontology\}$
$\{talk1\} \sqsubseteq \exists in.Session$
$\{talk1\} \sqsubseteq TargetTalk$
$\{talk1\} \sqsubseteq Talk4Dave$
$\{talk2\} \sqsubseteq Talk4Dave$
$\{talk2\} \sqsubseteq \exists recommend.\{Daivd\}$
$\{talk1\} \sqsubseteq \exists recommend.\{Daivd\}$

Figure 5: Erased TMS of $\mathcal{O}_0^1$ (1)

4. Check which new result of $Ans(\mathcal{O}_0^{n+1}(n+1), Q)$ can be entailed and add corresponding nodes and edges to $G_{\mathcal{O}_0^{n+1}(n+1)}$. In our example, new results $\{talk2\} \sqsubseteq \exists in.Session$, $\{talk2\} \sqsubseteq Talk4Dave$ and $\{talk2\} \sqsubseteq TargetTalk$ should be added.

As a example, applying the above procedure on our example $\mathcal{O}_0^2(2)$ will produce the following new TMS (Fig. 6). In this TMS the green nodes and edges are newly added while the others are preserved from $G_{\mathcal{O}_0^1(1)}$.



$\mathcal{O}_0^2(2)$
$ActiveTalk \sqsubseteq \exists in.Session$
$\{talk2\} \sqsubseteq ActiveTalk$
$ActiveTalk \sqcap Talk4Dave \sqsubseteq TargetTalk$
$\exists recommend.\{David\} \sqsubseteq Talk4Dave$
$hasTopic \circ interest \sqsubseteq recommend$
$\{ontology\} \sqsubseteq \exists interest.\{David\}$
$\{talk1\} \sqsubseteq \exists hasTopic.\{ontology\}$
$\{talk2\} \sqsubseteq \exists hasTopic.\{ontology\}$
$\{talk2\} \sqsubseteq \exists in.Session$
$\{talk2\} \sqsubseteq TargetTalk$
$\{talk1\} \sqsubseteq Talk4Dave$
$\{talk2\} \sqsubseteq Talk4Dave$
$\{talk2\} \sqsubseteq \exists recommend.\{Daivd\}$
$\{talk1\} \sqsubseteq \exists recommend.\{Daivd\}$

Figure 6: TMS of $\mathcal{O}_0^2$ (2) $G_{\mathcal{O}_0^2(2)}$

As we can see, the answer to David's question has been updated after the updating of ongoing events in the conference. During this procedure, intermediate results $\{talk2\} \sqsubseteq \exists recommend.\{Daivd\}$ and $\{talk2\} \sqsubseteq Talk4Dave$ have been preserved and reused to infer the new results, without being re-computed.

The way in which a TMS is constructed has significant impact on its structure and performance. A most straightforward way is to directly connect reasoning results with their justifications. However this approach contradicts with the intention of stream reasoning as it omits all intermediate results. This leads to potentially redundant computation of justifications and potentially redundant edges. For example, if $\alpha, \beta \in Ans(\mathcal{O}, Q)$ and $\mathcal{O} \models \beta$ only if $\mathcal{O} \models \alpha$, then α is needed to entail β and $J_{\mathcal{O}}(\alpha) \subseteq J_{\mathcal{O}}(\beta)$. Obviously, when computing $J_{\mathcal{O}}(\beta)$, the $J_{\mathcal{O}}(\alpha)$ should be computed at the same time, and all edges from axioms in $J_{\mathcal{O}}(\alpha)$ to β can be replaceable by a single edge from α to β.

To overcome the above drawbacks and improve efficiency, one can also develop a glass-box approach. The basic idea is to capture all the run-time states of a reasoner. For example, whenever a new belief is entailed from the ontology, we include it as a node of the TMS and connect it with the beliefs or assumptions we use to entail it, which should also be included as nodes if not included in previous reasoning steps. If an assumption is made, we include this assumption as a new node into the TMS. If it is withdrawn, we remove this assumption. All nodes derived from it, and all corresponding edges, should also be removed, etc. In this approach, the construction of the TMS does not affect the termination, soundness, completeness of the original reasoning algorithms. It does not change the reasoning results. These are all because the TMS does not interfere with the reasoning procedure. It does not require additional reasoning, and can be performed on-the-fly with reasoning.

Due to the fact that the completion-based algorithm of $\mathcal{EL}^{++}$ uses entailments as antecedents and consequents of rules, and requires no assumption in reasoning, it naturally fit with the construction of a TMS. And the glass-box approach for a completion-based algorithm can be described by the following algorithm **A-1**:

Algorithm A-1:
$GlassboxTMS(\mathcal{O}, Q)$
INPUT: an ontology $\mathcal{O}$, a reasoning request Q
OUTPUT: a directed graph $G = \langle N_{\mathcal{O}}, E_{\mathcal{O}} \rangle$

```
1: $N_{\mathcal{O}} := \mathcal{O}$, $E_{\mathcal{O}} := \{(\alpha, \alpha) | \alpha \in \mathcal{O}\}$
2: while a rule $R = \bigwedge_{i=1,2,\ldots,n} \alpha_i \rightarrow \bigwedge_{j=1,2,\ldots,m} \beta_j$ is executed for $Q$ do
3:     $N_{\mathcal{O}} := N_{\mathcal{O}} \cup \bigcup_{j=1,2,\ldots,m} \{\beta_j\}$
4:     $E_{\mathcal{O}} := E_{\mathcal{O}} \cup \bigcup_{i=1,2,\ldots,n, j=1,2,\ldots,m} \{(\alpha_i, \beta_j)\}$
5: return $\langle N_{\mathcal{O}}, E_{\mathcal{O}} \rangle$
```

It is not difficult to show that the constructed directed graph is a TMS of $\mathcal{O}$ containing all results in $Ans(\mathcal{O}, Q)$:

THEOREM 1. *For an ontology $\mathcal{O}$ and a reasoning request Q, $GlassboxTMS(\mathcal{O}, Q) = \langle N_{\mathcal{O}}, E_{\mathcal{O}} \rangle$ as computed by **A-1** is a TMS of $\mathcal{O}$ and $Ans(\mathcal{O}, Q) \subseteq N_{\mathcal{O}}$.*

TMS can thus be constructed and updated. First of all, given an $\mathcal{EL}^{++}$ ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, we internalize it into $\mathcal{O}' = (\mathcal{T}', \emptyset)$ and we initialise a TMS $G = \langle N_{\mathcal{T}'}, E_{\mathcal{T}'} \rangle$ as $N_{\mathcal{T}'} = \mathcal{T}'$ and $E_{\mathcal{T}'} = \{(\alpha, \beta) | \alpha \in \mathcal{A}, \beta \in \mathcal{T}'$ and β is internalised from $\alpha\} \cup \{(\alpha, \alpha) | \alpha \in \mathcal{T} \cup \mathcal{A}\}$. Then we normalise the TBox $\mathcal{T}'$ into its normal form $\mathcal{T}''$. Meanwhile we extend the TMS as $N_{\mathcal{T}'} = \mathcal{T}''$ and $E_{\mathcal{T}'} := E_{\mathcal{T}'} \sqsubseteq \{(\alpha, \beta) | \alpha \in \mathcal{T}', \beta \in \mathcal{T}''$ and β is normalised from $\alpha\}$. Then in reasoning, we apply completion rules to add new entailments into $\mathcal{T}''$, which is also the node set of the TMS. At the same time, we add edges between all antecedents and consequent into $E_{\mathcal{T}'}$. For example, for **R1**, we have $E_{\mathcal{T}'} := E_{\mathcal{T}'} \cup \{(X \sqsubseteq A, X \sqsubseteq B), (A \sqsubseteq B, X \sqsubseteq B)\}$. For **R5**, we have $E_{\mathcal{T}'} := E_{\mathcal{T}'} \cup \{(X \sqsubseteq \exists r.A, X \sqsubseteq \bot), (A \sqsubseteq \bot, X \sqsubseteq \bot)\}$

When no rule can be executed, the reasoning terminates and the TMS is constructed. When the TMS is updated from $\mathcal{O}_0^n$ (n) to $\mathcal{O}_0^{n+1}$ (n+1), the same procedure is applied on the previous TMS.

5. OPTIMISING TMS IN $\mathcal{EL}^{++}$

Although completion-based algorithms are suitable for stream reasoning with TMS. They could still unnecessarily consume some time and memory if not all intermediate results are contributing to answering the reasoning request. Taking $\mathcal{EL}^{++}$ algorithm as an example, it performs the **Classification** of a TBox by computing results of form $A \sqsubseteq B$ and $A \sqsubseteq \exists r.B$. The former is useful when computing the concept hierarchies of the ontology, or retrieving the types of individuals (in form of $\{x\} \sqsubseteq A$), while the later is of less use in such scenarios. For example, in Fig. 4 and Fig. 6, we can see that inference results $\{talk1\} \sqsubseteq \exists in.Session$ and $\{talk2\} \sqsubseteq \exists in.Session$ are not needed.

In order to address this issue, we further optimise completion-based algorithms. The key point is, instead of directly performing the forward-chaining reasoning procedure and apply all executable rules, we develop some backward-chaining control mechanism to determine which intermediate results will *NOT* be useful and thus they are not needed to be computed.

As we can see from Table 1, the intermediate results of the form $A \sqsubseteq \exists r.B$ are involved in the following rules:

1. **R3**, **R7** and **R8** use them to generate results of the same form.
2. **R4** and **R6** use them to generate new results of form $A \sqsubseteq B$.

This means that, all results of form $A \sqsubseteq \exists r.B$ will eventually contribute to inference of results of form $A \sqsubseteq B$ through **R4** and **R6**. If we know which axioms of form $A \sqsubseteq \exists r.B$ will (not) be used in **R4** and **R6**, we can inductively know which of them will (not) be used in the entire reasoning process to compute all results of form $A \sqsubseteq B$, without even executing any rule. Apparently, this is related to the role r in the entailment.

Given an internalised and normalised $\mathcal{EL}^{++}$ TBox $\mathcal{T}$, we assume that all role subsumption closures have been pre-computed [1]. We analyse the roles as follows:

In **R4:** If $X \sqsubseteq \exists r.A$, $A \sqsubseteq A'$, $\exists r.A' \sqsubseteq B$ then $X \sqsubseteq B$. It is obvious that antecedents of form $X \sqsubseteq \exists r.A$ is worth entailed only if there is some $\exists r.A' \sqsubseteq B \in \mathcal{T}$. Note that the later will not be computed from any rule and can only exist in the original TBox. Therefore for a role r, if there is no axiom of form $\exists r.A' \sqsubseteq B \in \mathcal{T}$, all results of form $X \sqsubseteq \exists r.A$ will not be useful in **R4**.

In **R6:** If $X, A \sqsubseteq \{a\}$, $X \leadsto_R A$ then $X \sqsubseteq A$. It is obvious that antecedent axioms of form $C_j \sqsubseteq \exists r_j.C_{j+1}$ is worth entailed only if there is some $X \sqsubseteq \{a\}$ entailed, which eventually requires there being some $\{a\}$ appearing on the RHS (Right Hand Side) of some GCI. Therefore, if there is no nominal $\{a\}$ on the RHS of any GCI, all results of form $X \sqsubseteq \exists r.A$ will not be used in **R6**.

From the above analysis, we define the classification-relevant roles as follows:

DEFINITION 3. ***(Classification-relevant role)*** *Given a normalised $\mathcal{EL}^{++}$ TBox $\mathcal{T}$, a role r is classification relevant (C-R for short) if r is on the LHS (left hand side) of some GCI, or there exists some GCI whose RHS (right hand side) is a nominal, or r is on the LHS of some RI whose RHS is a C-R role.*

The first two conditions correspond to **R4** and **R6**, respectively. The last condition corresponds to **R7** and **R8**. According to this definition, in our example, roles $hasTopic$, $interest$, $recommend$ are C-R, and role in is not. All C-R roles can be identified in polynomial time. They control which entailment $A \sqsubseteq \exists r.B$ will be useful in classification:

THEOREM 2. *When classifying an $\mathcal{EL}^{++}$ TBox $\mathcal{T}$ with **R1-8**, an entailed $A \sqsubseteq \exists r.B$ will contribute to execution of **R4** or **R6** only if r is a C-R role.*

In classification of an $\mathcal{EL}^{++}$ TBox, we computes intermediate results of form $A \sqsubseteq \exists r.B$ only if r is C-R. Thus the corresponding **R3**, **R7** and **R8** should be revised as follows:

- **R3'**: If $X \sqsubseteq A$, $A \sqsubseteq \exists r.B$ and r is C-R, then $X \sqsubseteq \exists r.B$;
- **R7'**: If $X \sqsubseteq \exists r.A$, r is C-R, $r \sqsubseteq s$ then $X \sqsubseteq \exists s.A$
- **R8'**: If $X \sqsubseteq \exists r_1.A$, $A \sqsubseteq \exists r_2.B$, $r_1 \circ r_2 \sqsubseteq r_3$, r_3 is C-R, then $X \sqsubseteq \exists r_3.B$

Such an optimisation is not only applicable on TMS construction. It is a generic optimisation on $\mathcal{EL}^{++}$ classification algorithm and preserves soundness and completeness of classification:

THEOREM 3. *For any normalised $\mathcal{EL}^{++}$ TBox $\mathcal{T}$, $A \sqsubseteq B$ can be inferred by **R1-8** iff $A \sqsubseteq B$ can be inferred by **R1-2, R3', R4-6, R7'-8'**.*

When constructing TMS with a glass-box approach, the TMS enjoys the same optimisation. The construction of TMS will follow the same rule as before. The fact that some role r is C-R is not necessary to be maintained as a node. For example, the optimised TMS of $\mathcal{O}_0^1$ (1) is illustrated in the Fig. 7:

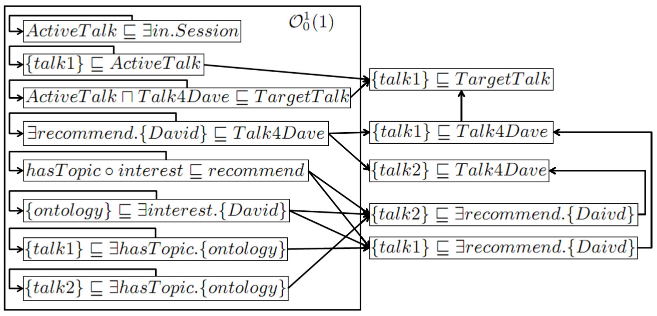

$\mathcal{O}_0^1(1)$
$ActiveTalk \sqsubseteq \exists in.Session$
$\{talk1\} \sqsubseteq ActiveTalk$
$ActiveTalk \sqcap Talk4Dave \sqsubseteq TargetTalk$
$\exists recommend.\{David\} \sqsubseteq Talk4Dave$
$hasTopic \circ interest \sqsubseteq recommend$
$\{ontology\} \sqsubseteq \exists interest.\{David\}$
$\{talk1\} \sqsubseteq \exists hasTopic.\{ontology\}$
$\{talk2\} \sqsubseteq \exists hasTopic.\{ontology\}$
$\{talk1\} \sqsubseteq TargetTalk$
$\{talk1\} \sqsubseteq Talk4Dave$
$\{talk2\} \sqsubseteq Talk4Dave$
$\{talk2\} \sqsubseteq \exists recommend.\{Daivd\}$
$\{talk1\} \sqsubseteq \exists recommend.\{Daivd\}$

Figure 7: Optimised TMS of $\mathcal{O}_0^1$ (1)

Such modifications will help eliminate unnecessary intermediate results and will not require additional reasoning. Therefore it should improve both the efficiency and scalability.

6. EXPERIMENTAL EVALUATION

We implemented the proposed approach and optimisations in our TrOWL [1] reasoner. We perform experiments to evaluate whether the optimisation presented in Sec.5 can improve the efficiency of TMS-enabled reasoning and reduce memory consumption, and furthermore how does such an optimised TMS-enabled $\mathcal{EL}^{++}$ stream reasoner perform in reality. Therefore, we first evaluate the optimisation, then the stream reasoning. All experiments were conducted in an environment of 64-bit Windows 7 Enterprise with 1.60 GHz CPU and 3G RAM allocated to JVM 1.6.0.07.

In this optimisation evaluation, our test cases are two ontologies generated from the Galen ontology [2]. Both of them are $\mathcal{EL}^{++}$ ontologies. The stats of these two ontologies are summarized in Table 2. Although they don't contain ABox, the optimisation evaluated here will have the similar effects on ABoxes.

In this evaluation we perform ontology classification to retrieve entailed subsumptions between all pairs of atomic concepts. For each of the ontologies, we classify it with two TMS-enabled $\mathcal{EL}^{++}$ reasoners. The one without optimisation implemented is called **TMS**. And the one with optimisation implemented is called **TMS-Opt**. For each reasoner, we measure the time used to classify the ontology. Due to the difficulty of precise measurements of the memory consumption, we calculate the number of intermediate results of the form $A \sqsubseteq \exists r.B$ for each reasoner. For **TMS-Opt** we further calculate the number of C-R roles. The results of our evaluation is shown in Table 2.

From results we can see that in both ontologies, not all roles are necessary for classification. With **TMS-Opt**, the number of unnecessary intermediate results ($\#Ext.$) is significantly reduced. In NotGalen$^-$, only 15.3% is computed. In FullGalen$^-$ only 39.8% is computed. With less unintended and unnecessary inference, the efficiency is improved (24% in NotGalen$^-$ and 44% in FullGalen$^-$). Also, the memory consumption of maintaining such inference should also be significantly reduced.

With the positive results on TMS optimisations, we evaluate its performance in stream reasoning. Due to the lack of stream reason-

[1]http://trowl.eu
[2]http://www.opengalen.org/

Table 2: Evaluation of optimisation. $|\mathcal{CN}|$, $|\mathcal{RN}|$, $\#GCI$ and $\#RI$ are numbers of atomic concepts, atomic roles, GCIs and RIs, respectively. $\#Sub.$ is the number of entailed atomic concept subsumptions. t is classification time (in seconds). $\#Ext.$ is the number of entailed intermediate results $A \sqsubseteq \exists r.B$. $\#C-R$ is the number of classification-relevant roles.

Ontology	$\|\mathcal{CN}\|$	$\|\mathcal{RN}\|$	$\#GCI$	$\#RI$	$\#Sub.$	TMS		TMS-Opt		
						t	$\#Ext.$	t	$\#Ext.$	$\#C-R$
NotGalen$^-$	2748	413	4348	442	30682	1,843	73156	1.407	11294	245
FullGalen$^-$	23141	950	37482	1015	475211	45.741	1687552	25.572	671595	749

ing benchmark, we generated our test data from the FullGalen$^-$ ontology and simulated streams. We randomly partition the ontology into 45 sub-ontologies of same size. We call them $\mathbf{K}_0, \mathbf{K}_1, \ldots, \mathbf{K}_{44}$. We construct an ontology stream $\mathcal{O}_0^{10}$ in a way that $\mathcal{O}_0^{10}(i) = \bigcup_{j=i,\ldots,i+34} \mathbf{K}_j$. Therefore for each $i = 0, \ldots, 9$, $\mathcal{O}_0^{10}(i+1) = \mathcal{O}_0^{10}(i) \setminus \mathbf{K}_i \cup \mathbf{K}_{i+35}$. Now we create a stream with 10 updates. In each update, 2.9% of the ontology is changed. Each snapshot contains about 30000 axioms. Each update changes about 855 axioms. For each snapshot $\mathcal{O}_0^{10}(i)$, we perform classification to compute the subsumption between all atomic concepts. The reasoner does not know which axioms will be updated in advance. We accomplish such a reasoning task with both the naive approach (re-do reasoning on $\mathcal{O}_0^{10}(i)$ from scratch) and the **TMS-Opt**. We record the time of both approaches to evaluate the effectiveness of the **TMS-Opt**. The results are summarised in the following table (Table 3).

Table 3: Evaluation Results. t_0 shows the time for the initial ontology. $Max(t_i)$, $Min(t_i)$ and $Ave(t_i)$ show the maximal, minimal and average time for the updated ontologies, respectively. Time unit is second.

Approach	t_0	$Max(t_i)$	$Min(t_i)$	$Ave(t_i)$
Naive	4.119	3.323	2.625	3.125
TMS-Opt	5.492	2.543	1.482	1.856

From the table we can see that, **TMS-Opt** took more time to classify the original ontology due to the construction of TMS. But in updating, **TMS-Opt** was significantly faster.

It is interesting to see how many percentages of the ontology can be updated while the TMS stream reasoning approach remains faster than the naive approach. To answer this question, we further increase to size of update to 3.33%, 5% and 10% of the original ontology, respectively. We perform classification on such ontology streams and calculate the relative time as $(Ave(t_i)_{Naive} - Ave(t_i)_{\textbf{TMS-Opt}})/Ave(t_i)_{Naive}$. Our experiments and calculation yield 75.1%, 86.0% and 96.5% relative time for the above relative size of updates, respectively. From these results, we can see that although TMS gradually loses its efficiency advantage to Naive approach when the relative volume of update increases, it still pays off when about 10% of ontology is changed. This figure is better than the maintenance program approach [15] and comparable to the fixed-window approach [6]. Considering the higher expressive power and difficulty of the test ontology FullGalen$^-$ and its large volume, our evaluation justifies the usability of our approach.

7. CONCLUSION

In this paper, we introduce Ontology Stream Management Systems (OSMS). We focus on how to provide stream reasoning services based on TMS for OSMS. We chose $\mathcal{EL}^{++}$, the logic underpinning the tractable OWL 2 EL language and developed a classification-relevant role optimisation for $\mathcal{EL}^{++}$ by controlling the kind of intermediate results that should be inferred. Evaluation on real world ontologies shows that our approach and optimisation work nicely in practice with relatively large volume of knowledge base and update. In the future, we would like to investigate stream reasoning in complete OWL 2 EL, QL and RL profiles, and even OWL 2 DL, as well as related optimisations.

Acknowledgement: This work is partially funded by the RCUK dot.rural and the EU K-Drive projects.

8. REFERENCES

[1] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ Envelope. In *Proceedings IJCAI-05*, 2005.

[2] F. Baader, S. Brandt, and C. Lutz. Pushing the EL Envelope Further. In K. Clark and P. F. Patel-Schneider, editors, *In OWLED-2008*, 2008.

[3] F. Baader, C. Lutz, and B. Suntisrivaraporn. Is Tractable Reasoning in Extensions of the Description Logic EL Useful in Practice? In *Proceedings of the 2005 International Workshop on Methods for Modalities (M4M-05)*, 2005.

[4] F. Baader and B. Suntisrivaraporn. Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In *Proceedings of the 3rd Knowledge Representation in Medicine (KR-MED'08): Representing and Sharing Knowledge Using SNOMED*, volume 410 of *CEUR-WS*, 2008.

[5] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. C-sparql: Sparql for continuous querying. In *WWW2009*, 2009.

[6] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. Incremental reasoning on streams and rich background knowledge. In *ESWC2010*, 2010.

[7] A. Bolles, M. Grawunder, and J. Jacobi. Streaming sparql extending sparql to process data streams. In *ESWC08*, 2008.

[8] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[9] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *SIGMOD '93*, 1993.

[10] Z. Huang and H. Stuckenschmidt. Reasoning with multi-version ontologies: A temporal logic approach. In *In Proceeding of ISWC2005*, 2005.

[11] M. Luther and S. Bohm. Situation-Aware Mobility: An Application for Stream Reasoning. In *in Proc. of 1st International Workshop on Stream Reasoning (SR2009)*, 2009.

[12] B. Parsia, C. Halaschek-wiener, and E. Sirin. E.s.: Towards incremental reasoning through updates. In *in OWL DL. In: Proc. WWW-2006. (2006)*, 2006.

[13] Y. Ren, J. Z. Pan, and Y. Zhao. Towards Scalable Reasoning on Ontology Streams via Syntactic Approximation. In *the Proc. of IWOD2010*, 2010.

[14] R. Volz, S. Staab, and B. Motik. Incementally maintaining materializations of ontologies stored in logic databases. *In Journal of Data Semantics II, LNCS, Vol 3360*, 2:1–34, 2005.