Ontological Stream Reasoning via Syntactic Approximation

Yuan Ren, Jeff Z. Pan, and Yuting Zhao

Dept. of Computing Science
University of Aberdeen
Aberdeen, UK

Abstract. Traditionally Description Logic (DL)-based ontologies are only used to capture static knowledge. Recently the fast development of the Semantic Web and its evolving data raises the requirements of reasoning services on dynamic knowledge streams. In this paper, we present an efficient ontology stream reasoning approach which combines the delete and re-derive algorithm and the syntactic approximation to guarantee soundness and tractability. Compared with existing works, it allows stream reasoning for ontologies in very expressive language profiles and requires no prior knowledge about the streams.

1 Introduction

Traditionally Description Logic (DL)-based ontologies are only used to capture static knowledge. That is to say, the knowledge base is assumed to be permanent. However the dynamics of knowledge is becoming more and more frequent in various applications, including real time reasoning in ontology editors, sensor data streams, semantic web content updates and mobile semantic web [13, 12]. In these cases, in reasoning we should consider the erasure of old knowledge and adding of new knowledge. This raises the requirements of stream reasoning, in which people want to do reasoning with changing knowledge [17, 18].

In [19] a delete and re-derive (DRed) approach is adopted from the stream data processing in relational database. When the updates of knowledge base occur, the stream reasoner first erases all the entailments that rely on the removed axioms, then re-do the reasoning over the remaining axioms and added axioms to entail new results. Later, [6] realized that if the time window of stream is fixed, we can use a time tag to annotate each entailed axiom, so that when the time of expiration comes, the stream reasoner immediately knows whether an entailment can be erased. Similarly, when a new entailment is derived, the stream reasoner immediately knows when it will expire. However, the applicability of the later approach relies on the prior knowledge of fixed stream time window. Further more, both of these works still focus on rather simple languages such as RDF and RDFs and relatively small knowledge base.

In this paper, we propose a novel approach which applies syntactic approximation to reduce the reasoning complexity. Instead of trivially using syntactic approximate reasoning to compute entailments, we extend it with the traceability information among

entailments, so that when erasing axioms, we can efficiently detect the entailments impacted by such erasure. When adding axioms, we augment the syntactic approximation with the incremental reasoning facility of $\mathcal{EL}^+$/$\mathcal{EL}^{++}$ [16].

The remainder of the paper is organized as follows: In Sec.2 we introduce background knowledge of DL, ontology and the syntactic approximation. In Sec.3 we review existing works on stream reasoning and propose the big picture of our approach. The next two sections present our approach. First in Sec.4 we extend the syntactic approximation so that we maintain the traceability relations among them when we derive entailments. In Sec.5 we utilize the traceability information to efficiently analyze the impact of erasure of axioms. And further apply incremental reasoning technique to deal with adding of axioms. Our proposal is implemented and evaluated in Sec.6. In Sec.7 we discuss two possible extensions of our approach. Sec.8 summaries the paper.

2 Background

2.1 DL and Ontology

DL is a family of logic-based knowledge representation formalisms. They vary in language expressive power and computational complexities. In this paper we focus on the DL $\mathcal{SROIQ}$ and $\mathcal{EL}^{++}$. For the sake of conciseness, we highlight the parts of syntax and semantics that are of special interest in our approximate reasoning approach. Complete specifications of these two languages can be found in [10] and [1, 2], respectively.

In DL, concept and role expressions are composed with language constructs. Let $\top$ be the top concept, $\bot$ the bottom concept, A an atomic concept, n an integer number, a an individual, r an atomic role, in $\mathcal{SROIQ}$, concept expressions C, D can be inductively composed with the following constructs: $\top \mid \bot \mid A \mid C \sqcap D \mid \exists R.C \mid \{a\} \mid \neg C \mid \geq nR.C \mid \exists R.Self$ where R is a role expression which is either r or an inverse role R^-. And $\exists R.Self$ is a self-restriction.

Conventionally, $C \sqcup D$, $\forall R.C$ and $\leq nR.C$ are used to abbreviate $\neg(\neg C \sqcap \neg D)$, $\neg\exists R.\neg C$ and $\neg \geq (n+1)R.C$, respectively. $\{a_1, a_2, \ldots, a_n\}$ can be regarded as abbreviation of $\{a_1\} \sqcup \{a_2\} \sqcup \ldots \sqcup \{a_n\}$. Without loss of generality, in what follows, we assume all the concepts to be in their unique *negated normal forms* (NNF)[1] and use ~C to denote the NNF of $\neg C$. We also call $\top, \bot, A, \{a\}$ *basic concepts* because they are not composed by other concepts or roles. Given a knowledge base Σ (an ontology or a TBox, or an ABox), we use CN_Σ (RN_Σ) to denote the set of basic concepts (atomic roles) in Σ.

$\mathcal{EL}^{++}$ supports $\top \mid \bot \mid A \mid C \sqcap D \mid \exists r.C \mid \{a\}$, where r is an atomic role.

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. Both $\mathcal{SROIQ}$ and $\mathcal{EL}^{++}$ support concept inclusion axioms (CIs, e.g. $C \sqsubseteq D$) and role inclusion axioms (RIs, e.g. $r \sqsubseteq s$, $r_1 \circ \ldots \circ r_n \sqsubseteq s$). $\mathcal{SROIQ}$ supports also other axioms such as functionality of roles (e.g. $func(r)$), inverse roles (e.g. $inverseRole(r, s)$), etc. If $C \sqsubseteq D$ and $D \sqsubseteq C$, we write $C \equiv D$. If C is non-atomic, $C \sqsubseteq D$ is a general concept inclusion axiom (GCI). An ABox is a set of assertion axioms. Both $\mathcal{SROIQ}$ and $\mathcal{EL}^{++}$ support the concept assertion axioms,

[1] An $\mathcal{SROIQ}$ concept is in NNF *iff* negation is applied only to atomic concepts, nominals or Self-restriction. NNF of a given concept can be computed in linear time [9].

e.g. $a : C$, role assertion axioms, e.g. $(a, b) : R$, individual equality, e.g. $a = b$ and inequality, e.g. $a \neq b$.

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 $\mathcal{O}$.

2.2 Syntactic Approximation

Syntactic approximation [14, 15] is an approximate reasoning technique that reduces DL reasoning in $\mathcal{SROIQ}$ to $\mathcal{EL}^{++}$. The reasoning complexity is thus reduced from 2NEXPTIME-Complete to PTIME-Complete while the results are guaranteed correct. In this subsection, we briefly recall the syntactic approximation technique and its features. For more details about proofs, readers are referred to [14, 15]. In later sections, we will extend the current approach to support required closed world reasoning services.

The idea of syntactic approximation from $\mathcal{SROIQ}$ and $\mathcal{EL}^{++}$ is to first encode non-$\mathcal{EL}^{++}$ expressions with fresh names, then maintain their semantics with additional axioms and separate data structures. For example, complement relations between an named concept A and the new name, e.g. nA, assigned to its complement $\neg A$ are maintained in the *Complement Table* (CT). In reasoning phase, additional completion rules will be used to partially recover the semantics.

In approximation, we only consider concepts corresponding to the particular TBox in question. We use the notion *term* to refer to these "interesting" concept expressions. More precisely, a term is: (i) a concept expression in any axiom, or (ii) a singleton of any individual, or (iii) the complement of a term, or (iv) the syntactic sub-expression of a term. In order to represent all these terms and role expressions that will be used in $\mathcal{EL}^{++}$ reasoning, we first assign names to them.

Definition 1. ***(Name Assignment)*** *Given S a set of concept expressions, E a set of (negative) role expressions, a* name assignment *fn is a function as for each $C \in S$ ($R \in E$), $fn(C) = C$ ($fn(R) = R$) if C is a basic concept (R is named), otherwise $fn(C)$ ($fn(R)$) is a fresh name.*

Now we can transform ontologies into $\mathcal{EL}^{++}$ with additional data structures by the following definition.

Definition 2. ***($\mathcal{EL}^{++}_{\mathcal{CQ}}$ Transformation)*** *Given an Ontology $\mathcal{O}$ and a name assignment fn, its $\mathcal{EL}^{++}_{\mathcal{CQ}}$* transformation *$A_{fn,\mathcal{EL}^{++}_{\mathcal{CQ}}}(\mathcal{O})$ is a triple $(\mathcal{T}, CT, QT)$ constructed as follows:*

1. *$\mathcal{T}, CT$ and QT are all initialized to $\emptyset$.*
2. *for each $C \sqsubseteq D$ ($C \equiv D$) in $\mathcal{O}$, $\mathcal{T} = \mathcal{T} \cup \{fn(C) \sqsubseteq fn(D)\}$ ($\mathcal{T} = \mathcal{T} \cup \{fn(C) \equiv fn(D)\}$).*
3. *for each $\mathcal{EL}^{++}$ role axiom $\beta \in \mathcal{O}$, add $\beta_{[R/fn(R)]}$ into $\mathcal{T}$.*
4. *for each $a : C \in \mathcal{O}$, $\mathcal{T} = \mathcal{T} \cup \{\{a\} \sqsubseteq fn(C)\}$.*
5. *for each $(a, b) : R \in \mathcal{O}$, $\mathcal{T} = \mathcal{T} \cup \{\{a\} \sqsubseteq \exists fn(R).\{b\}, \{b\} \sqsubseteq \exists fn(Inv(R)).\{a\}\}$.*
6. *for each $a \doteq b \in \mathcal{O}$, $\mathcal{T} = \mathcal{T} \cup \{\{a\} \equiv \{b\}\}$.*
7. *for each $a \neq b \in \mathcal{O}$, $\mathcal{T} = \mathcal{T} \cup \{\{a\} \sqsubseteq \neg\{b\}\}$*
8. *for each term C in $\mathcal{T}$, $CT = CT \cup \{(fn(C), fn(\sim C))\}$, and*

(a) *if* C *is the form* $C_1 \sqcap \ldots \sqcap C_n$, *then* $\mathcal{T} = \mathcal{T} \cup \{fn(C) \equiv fn(C_1) \sqcap \ldots \sqcap fn(C_n)\}$,
(b) *if* C *is the form* $\exists r.D$, *then* $\mathcal{T} = \mathcal{T} \cup \{fn(C) \equiv \exists r.fn(D)\}$,
(c) *if* C *is the form* $\exists R.Self$, *then* $T = T \cup \{fn(C) \equiv \exists fn(R).Self\}$
(d) *if* C *is the form* $\geq nR.D$, *then*
  i. *if* $n = 0$, $\mathcal{T} = \mathcal{T} \cup \{\top \sqsubseteq fn(C)\}$
  ii. *if* $n = 1$, $\mathcal{T} = \mathcal{T} \cup \{fn(C) \equiv \exists fn(R).fn(D)\}$
  iii. *otherwise,* $\mathcal{T} = \mathcal{T} \cup \{fn(C) \equiv fn(D)^{fn(R),n}\}$, *and* $QT = QT \cup \{(fn(C), fn(R), n)\}$.
(e) *otherwise* $\mathcal{T} = \mathcal{T} \cup \{fn(C) \sqsubseteq \top\}$.

9. *for each pair of names* A *and* r, *if there exist* $(A, r, i_1), (A, r, i_2), \ldots, (A, r, i_n) \in QT$ *with* $i_1 < i_2 < \ldots < i_n$, $\mathcal{T} = \mathcal{T} \cup \{A^{r,i_n} \sqsubseteq A^{r,i_{n-1}}, \ldots, A^{r,i_2} \sqsubseteq A^{r,i_1}, A^{r,i_1} \sqsubseteq \exists r.A\}$

Step 2 rewrites all the concept axioms; Step 3 preserves all the $\mathcal{EL}^{++}$ role axioms; Step 4 to 7 rewrite all the ABox axioms and internalize them into the approximated TBox; Step 8 defines terms and constructs the complement table CT and cardinality table QT; Particularly, in step 8.(d), $fn(D)^{fn(R),n}$ is a fresh name. Obviously, this is unique for a given tuple of D, R and n. We call them *cardinality names*. Similarly, $\leq nR.D$ will be approximated via the approximation of its complement $\geq (n+1)R.D$. In step 9, for each pair of name assignment A, r in T, a subsumption chain is added into T because $\geq i_n r.A \sqsubseteq \ldots \sqsubseteq \geq i_2 r.A \sqsubseteq \geq i_1 r.A \sqsubseteq \exists r.A$.

We call this procedure an $\mathcal{EL}^{++}_{\mathcal{CQ}}$ *approximation*.The $\mathcal{EL}^{++}_{\mathcal{CQ}}$ approximation approximates an ontology into an $\mathcal{EL}^{++}$ ontology ([14, Proposition 1]) with a table maintaining the complementary relations and another table maintaining the cardinality relations. This approximation can be computed in linear time ([14, Proposition 2]).

Given an $\mathcal{EL}^{++}_{\mathcal{CQ}}$ transformation $(\mathcal{T}, CT, QT)$, we normalize axioms of form $C \sqsubseteq D_1 \sqcap \ldots \sqcap D_n$ into $C \sqsubseteq D_1, \ldots, C \sqsubseteq D_n$, and recursively normalize role chain $r_1 \circ \ldots \circ r_n \sqsubseteq s$ with $n > 2$ into $r_1 \circ \ldots \circ r_{n-1} \sqsubseteq u$ and $u \circ r_n \sqsubseteq s$. Because C, D_i are basic concepts, this procedure can be done in linear time. In the following, we assume $\mathcal{T}$ to be always normalized. For convenience, we use a *complement function* $fc : CN_{\mathcal{T}} \mapsto CN_{\mathcal{T}}$ as: for each $A \in CN_{\mathcal{T}}$, $fc(A) = B$ if $(A, B) \in CT$. Note that if A is a cardinality name, then it does not have a complement. In what follows, when applying $fc(A)$ we always assume that A is not a cardinality name but a assigned name.

With the normalized approximation, the reasoning can be realized by extending $\mathcal{EL}^{++}$ completion rules with support for the CT and QT. Given an ontology $\mathcal{EL}^{++}_{\mathcal{CQ}}$ transformation $(\mathcal{T}, CT, QT)$, the completion rules will compute, for each basic concept A, a subsumer set $S(A) \subseteq CN_{\mathcal{T}}$ such that if $B \in S(A)$ then $A \sqsubseteq B$, and for each named role r, a relation set $R(r) \subseteq CN_{\mathcal{T}} \times CN_{\mathcal{T}}$. For each basic concept A, $S(A)$ is initialized to $\{A, \top\}$ and for each named role r, $R(r)$ is initialized to $\emptyset$. An example of such rules can is illustrated as follows:

$$\text{If } A \in S(B) \text{ and } fc(B) \notin S(fc(A)), \text{ then } S(fc(A)) := S(fc(A)) \cup \{fc(B)\}$$

This rule asserts the reverse subsumption between concepts to supplement the absence of negation, i.e. $A \sqsubseteq B \rightarrow \sim A \sqsubseteq \sim B$.

The whole set of rules (**R1-R16**) can be found in [14]. Reasoning with them is tractable and sound ([14, Theorem 1 and 2]).

As in classical reasoning, unsatisfiability checking of a concept C can be reduced to entailment checking of $C \sqsubseteq \bot$; ontology inconsistency checking can be reduced to entailment checking of $\top \sqsubseteq \bot$ or $\{a\} \sqsubseteq \bot$.

3 Technical Motivation

In this section, we present a technical definition of stream reasoning, then show its significance and difficulty by analyzing exisiting works.

3.1 Stream Reasoning

As we introduced in early sections, a stream can be realized by updates on an ontology. Such updates consist of erasure of existing axioms and adding of new axioms.

Definition 3. *An ontology stream is a sequence of ontologies $\mathcal{O}(0), \mathcal{O}(1), \ldots$ constructed as follows:*

1. *$\mathcal{O}(0)$ is an initial ontology;*
2. *$\mathcal{O}(i+1) = \mathcal{O}(i) \setminus Er(i) \cup Ad(i)$.*

where $Er(i) \subseteq \mathcal{O}(i)$ makes up a sequence of erased axioms. $Ad(i)$ makes up a sequence of added axioms.

The change from $\mathcal{O}(i)$ to $\mathcal{O}(i+1)$ is an update. It's important to note that the sequences of erased axioms and added axioms are not necessarily known to reasoners in advance, i.e. when computing $Ans(\mathcal{O}(i), q)$, a stream reasoner does not necessarily know $Er(j)$ and $Ad(j)$ if $j \geq i$.

For a stream $\mathcal{O}(0), \mathcal{O}(1), \ldots$ and a reasoning problem q, applications ask for reasoning answers of q on $\mathcal{O}(i)$, denoted by $Ans(\mathcal{O}(i), q)$. Such answers can be directly computed on $\mathcal{O}(i)$, i.e. $Ans(\mathcal{O}(i), q) = f(\mathcal{O}(i), q)$, which means the answer is a function of the $\mathcal{O}(i)$ and q. We call this the *naive approach*.

More interestingly, people would like to compute $Ans(\mathcal{O}(i+1), q)$ based on $Ans(\mathcal{O}(i), q)$ so that partial results can be reused, i.e. $Ans(\mathcal{O}(i+1), q) = g(Ans(\mathcal{O}(i), 1), q, Er(i), Ad(i))$, which means the answer is a function of $Ans(\mathcal{O}(i+1)), q, Er(i)$ and $Ad(i)$. Note that, when $Er(i) = \emptyset$ for all i in the stream, stream reasoning is reduced to incremental reasoning because no axiom is erased. When $Ad(i) = \emptyset$ for all i in the stream, stream reasoning is reduced to incremental revision because no axiom is added.

3.2 Existing Works

There have been many works regarding querying streaming data on the semantic web. [7] proposed an extension of SPARQL to process data streams. Similarly, C-SPARQL [5, 4], another extension of SPARQL can continuously query from a RDF knowledge base.

More related works focus on continuously and incrementally updating and materializing ontological knowledge bases. [19] adopted the Delete and Re-derive (DRed) algorithm [8] 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. This approach maintains ontologies in logical databases and update them with logical programs thus it applied to ontology languages that can be translated to Datalog programs, such as RDF(S) and DLP. Also, the relations among asserted axioms and their consequences have to be maintained by additional rules so that when the asserted axioms are erased it is easy to final their consequences. This makes the maintenance program substantially large than the original program.

Similar idea was used in [6] with an additional assumption that the time window of stream is fixed and known to the stream reasoner. So that it is no longer needed to maintain the relations among asserted axioms and their consequences. Instead, the relation between a consequence and a time point should be maintained so that when the time comes, the stream reasoner expires corresponding consequences. However, this still requires updating expiration time of existing entailments when new facts are asserted into the ontology, which ends up with computing all possible justifications for each entailment, making it difficult to apply this approach to more expressive languages than RDF(S). Furthermore, if the time window is not known to the stream reasoner, this approach can not work.

3.3 Our Approach

Compared with existing works, our approach will have the following differences:

1. We apply syntactic approximation to stream reasoning. This enables stream reasoning for more expressive ontology languages.
2. Both asserted axioms and their consequences are entailed by the current ontology. When doing reasoning, we can keep traceability relations between the deriving facts and the derived facts. Such relations can be easily used to find impacted entailments in deletion.
3. When dealing with newly added axioms, we incrementally compute new entailments. Since we approximate ontologies to $\mathcal{EL}^{++}$, we can use its tractable incremental reasoning facility [16].

In the next sections, we first extend syntactic approximation to maintain traceability information on the fly, then use such information in updating ontologies.

4 Augmenting Syntactic Approximation with Traceability Information

In this section, we investigate how to further extend our proposed syntactic approximation to maintain the traceability among entailments.

The basic idea is to construct a directed traceability graph, whose nodes are entailments, whose edges are dependency between entailments such that if there exists an edge from entailment e_1 to entailment e_2 then e_2 is derived from e_1 (together with other entailments). It is obvious that a node can have multiple inbound edges if the entailment is derived from multiple other entailments. Asserted axioms and tautology

axioms do not have any inbound edges or only have inbound edges from themselves. In case of syntactic approximation, the traceability is established in two phases: the first phase is on the approximation and normalization (Sec 4.1), while the second phase is on reasoning (Sec 4.2).

4.1 Traced Approximation and Normalization

In the approximation and normalization phase, the input is a given ontology $\mathcal{O}$, while the output is an extended syntactic transformation $(\mathcal{T}, CT, QT)$. As introduced in Sec 2, in $(\mathcal{T}, CT, QT)$, elements of CT and QT are used to represent the semantics of some new names in $\mathcal{T}$. Therefore we only need to get trace of axioms in $\mathcal{T}$. Such traceability are maintained in a *Traceability Graph* $TG = (N, E)$. An element $e \in N$ is an entailment of $\mathcal{O}$ and $(\mathcal{T}, CT, QT)$ and $E \subseteq N \times N$ represents the derivation among entailments. This is realized by the following definition:

Definition 4. ***(Traced Syntactic Transformation)*** *Given an ontology $\mathcal{O}$ with $A_{fn,\mathcal{EL}^{++}_{CQ}}(\mathcal{O}) = (\mathcal{T}', CT', QT')$, then its* traced syntactic transformation *$tA_{fn,\mathcal{EL}^{++}_{CQ}}(\mathcal{O})$ is a four-tuple $(\mathcal{T}, CT, QT, TG)$, $TG = (N, E)$ constructed as follows:*

1. *$\mathcal{T} = \mathcal{T}'$, $CT = CT'$ and $QT = QT'$.*
2. *$N = \mathcal{O}$ and $E = \emptyset$.*
3. *for each $C \equiv D \in \mathcal{O}$, $N = N \cup \{fn(C) \sqsubseteq fn(D), fn(D) \sqsubseteq fn(C)\}$ and $E = E \cup \{(C \equiv D, fn(C) \sqsubseteq fn(D)), (C \equiv D, fn(D) \sqsubseteq fn(C))\}$,*
4. *for each $a \doteq b \in \mathcal{O}$, if $\{a\} \sqsubseteq \{b\}, \{b\} \sqsubseteq \{a\} \notin N$ then $N = N \cup \{\{a\} \sqsubseteq \{b\}, \{b\} \sqsubseteq \{a\}\}$ and $E = E \cup \{(a \doteq b, \{a\} \sqsubseteq \{b\}), (a \doteq b, \{b\} \sqsubseteq \{a\})\}$*
5. *for each $C \sqsubseteq D \in \mathcal{O}$, if $fn(C) \sqsubseteq fn(D) \notin N$ then $N = N \cup \{fn(C) \sqsubseteq fn(D)\}$ and $E = E \cup \{(C \sqsubseteq D, fn(C) \sqsubseteq fn(D))\}$*
6. *for each $a : C \in \mathcal{O}$, if $\{a\} \sqsubseteq fn(C) \notin N$ then $N = N \cup \{\{a\} \sqsubseteq fn(C)\}$ and $E = E \cup \{(a : C, \{a\} \sqsubseteq fn(C))\}$,*
7. *for each $(a, b) : R \in \mathcal{O}$, $N = N \cup \{\{a\} \sqsubseteq \exists fn(R).\{b\}, \{b\} \sqsubseteq \exists fn(Inv(R)).\{a\}\}$ and $E = E \cup \{((a, b) : R, \{a\} \sqsubseteq \exists fn(R).\{b\}), ((a, b) : R, \{b\} \sqsubseteq \exists fn(Inv(R)).\{a\})\}$*
8. *for each $a \neq b \in \mathcal{O}$, if $\{a\} \sqsubseteq fn(\neg\{b\}) \notin N$, then $N = N \cup \{\{a\} \sqsubseteq fn(\neg\{b\})\}$ and $E = E \cup \{(a \neq b, \{a\} \sqsubseteq fn(\neg\{b\}))\}$.*
9. *for each $R \equiv S \in \mathcal{O}$, $N = N \cup \{fn(R) \sqsubseteq fn(S), fn(S) \sqsubseteq fn(R)\}$ and $E = E \cup \{(R \equiv S, fn(R) \sqsubseteq fn(S)), (R \equiv S, fn(S) \sqsubseteq fn(R))\}$,*
10. *for each $R \sqsubseteq S \in \mathcal{O}$, if $fn(R) \sqsubseteq fn(S) \notin N$ then $N = N \cup \{fn(R) \sqsubseteq fn(S)\}$ and $E = E \cup \{(R \sqsubseteq S, fn(R) \sqsubseteq fn(S))\}$,*
11. *for each $R_1 \circ R_2 \sqsubseteq S \in \mathcal{O}$, $fn(R_1) \circ fn(R_2) \sqsubseteq fn(S) \notin N$ and $E = E \cup \{(\{R_1 \circ R_2 \sqsubseteq S\}, fn(R_1) \circ fn(R_2) \sqsubseteq fn(S))\}$,*
12. *for each axiom β of the form $R_1 \circ \ldots \circ R_n \sqsubseteq S \in \mathcal{O}$ with $n > 2$, $N = N \cup \{fn(R_1) \circ fn(R_2) \sqsubseteq u_{n-2}, u_1 \circ fn(R_n) \sqsubseteq fn(S)\} \cup \bigcup_{2 \leq k \leq n-2} \{u_k \circ fn(R_{n+1-k}) \sqsubseteq u_{k-1}\}$ $E = E \cup \{(\beta, fn(R_1) \circ fn(R_2) \sqsubseteq u_{n-2})\} \cup \bigcup_{2 \leq k \leq n-2} \{(\beta, u_k \circ fn(R_{n+1-k}) \sqsubseteq u_{k-1})\} \cup \{(\beta, u_1 \circ fn(R_n) \sqsubseteq fn(S))\}$*

The above definition creates traceability information for $\mathcal{T}$ when approximation and normalization are performed.

1. Step-3 and 5 deal with GCIs.
2. Step-4, 6 to 8 deal with internalized ABox axioms.
3. Step 9 to 12 deal with RIs. Note that in Step 3, 4 and 9, equivalent concept/role axioms are normalized, because the completion rules work on normalized axioms.
4. Similarly, in Step-12, role chain axioms are normalized. More precisely, an axiom $R_1 \circ \ldots \circ R_n \sqsubseteq S$ with $n > 2$ is approximated and then normalized in to axioms $fn(R_1) \circ fn(R_2) \sqsubseteq u_{n-2}, u_{n-2} \circ fn(R_3) \sqsubseteq u_{n-3}, \ldots, u_2 \circ fn(R_{n-1}) \sqsubseteq u_1, u_1 \circ fn(R_n) \sqsubseteq fn(S)$, where u_i are fresh new names. Their counter-parts, with the inverse role of u_i are also maintained. And all these axioms have their traceability relation with $R_1 \circ \ldots \circ R_n \sqsubseteq S$ maintained in TG. It is obvious that such normalization yields the same results as the normalization we mentioned in Sec.2.

It is also important to make Step-3 before 4, make 4 before 5 and 6, and make Step-9 before the other role axioms , because there might be duplications in approximating axioms. In this case we only maintain one of them to maintain tractability of the solution. For example, in $\{\{a\} \equiv A, \{a\} \sqsubseteq A, a : A\}$ the same approximated axiom $\{a\} \sqsubseteq A$ has three different asserted axioms. According to Def.4 only $(\{a\} \equiv A, \{a\} \sqsubseteq A)$ will be maintained in TG.

Proposition 1. *For an ontology $\mathcal{O}$, its traced syntactic transformation $tA_{fn,\mathcal{EL}^{++}_{CQ}}(\mathcal{O}) = (\mathcal{T}, CT, QT, TG)$ can be computed in linear time.*

According to the above proposition and Def.4, given a traced syntactic transformation $(\mathcal{T}, CT, QT, TG)$, all the axioms in $\mathcal{T}$ generated by Step-2 and 3 in Def.2 have been normalized traced to their asserted axioms in $\mathcal{O}$. The other axioms, i.e. those generated by Step-8 and 9 of Def.2 do not have traceability information because they are actually tautologies and do not rely on axioms in $\mathcal{O}$. These axioms will still need to be normalized and the normalized axioms should be added into TG as nodes. In what follows, we always assume $\mathcal{T}$ to be completely normalized.

4.2 Traced Reasoning

In the reasoning, the input is a traced syntactic transformation $(\mathcal{T}, CT, QT, TG)$, and the output are the subsumer sets of basic concepts and relation sets of atomic roles. We need to maintain the traceability information regarding elements of these sets. For example, $B \in S(A)$ only if $A \sqsubseteq B$ can be inferred, then we should add $A \sqsubseteq B$ as a node in the TG and connect it to other entailments.

In addition, the computation of subsumption closure of roles are implicitly treated in the original reasoning paradigm. Here we make them explicit and maintain the traceability information as well. We use $r \sqsubseteq_* s$ to denote that $r \sqsubseteq s$ can be derived. In initialization of the reasoning, we have $r \sqsubseteq_* r$ for every $r \in RN_{\mathcal{T}}$. Obviously $r \sqsubseteq_* r$ should be added into TG without inbound edges.

To this end, we can maintain the traceability of inferences results when applying the completion rules. For each rule, the descendent axiom should be added into the TG as a new node, and an edge should be created from each of the antecedent axioms to the descendent axiom. For example, suppose the traceability graph $TG = (N, E)$, the traced completion rule for role subsumption is as follows:

If $r_1 \sqsubseteq_* r_2$, $r_2 \sqsubseteq r_3$ and $r_1 \sqsubseteq r_3$ not inferred then infer $r_1 \sqsubseteq_* r_3, N = N \cup \{r_1 \sqsubseteq_* r_3\}$ and $E = E \cup \{(r_1 \sqsubseteq_* r_2, r_1 \sqsubseteq_* r_3), (r_2 \sqsubseteq r_3, r_1 \sqsubseteq_* r_3)\}$

Another example is the rule we showed at the end of Sec.2.2:

If $A \in S(B)$ and $fc(B) \notin S(fc(A))$ then $S(fc(A)) := S(fc(A)) \cup \{fc(B)\}$, $N = N \cup \{fc(A) \sqsubseteq fc(B)\}$ and $TT = TT \cup \{(B \sqsubseteq A, fc(A) \sqsubseteq fc(B))\}$

The other rules can be extended in a similar way. A useful optimization when generating traceability edges is that, if a node is not reachable from any asserted axiom, we do not need to generate any outbound edges from it. This is because when erasing asserted axioms from the ontology, such a node will not be impacted. Thus, an entailment derived from it will be impacted only if it is also connected to another impacted node.

Obviously, the traced completion rules have the same time complexity and reasoning quality as the original rules:

Theorem 1. *For any ontology $\mathcal{O}$ and its traced syntactic transformation $(\mathcal{T}, CT, QT, TG)$, TBox reasoning by traced completion rules will terminate in polynomial time w.r.t. $|CN_{\mathcal{O}}| + |RN_{\mathcal{O}}|$ and is soundness guaranteed.*

Because the syntactic approximation computes the subsumption between all named concepts in "one go", thus the $tA_{fn,\mathcal{EL}^{++}_{\mathcal{CQ}}}(\mathcal{O})$, especially the traceability graph $TG = (N, E)$, can be regarded as a by-product of $Ans(\mathcal{O}, q)$ for any ontology $\mathcal{O}$ and any reasoning problem q. If the reasoning problem is to classify the ontology, then $Ans(\mathcal{O}, q) \subseteq N$ because N contains all the entailed concept subsumptions.

In previous works [11], similar techniques are used to compute justifications in a glass-box manner. The major differences are:

1. In justification computation, the traceability information is used in a backward manner, i.e. given an entailment, finding all asserted axioms that derive it. In stream reasoning, as we will show, the traceability information is used in a forward manner, i.e. given an asserted axiom, finding all entailments impacted by it.
2. In justification computation, traceability information is recursively collapsed, i.e. the intermediate entailments are not part of the justification, only the original axioms are. In stream reasoning, the intermediate entailments also count because they are also part of the reasoning results.

5 Stream Reasoning with Traced Syntactic Approximation

We follow the idea of DRed to split the updates of ontologies into two steps: erasing and adding.

5.1 Erasing Axioms from Ontology

In the erasing step, the traceability graph provides a convenient way to find all entailments impacted by the erased axioms. More precisely, suppose $tA_{fn,\mathcal{EL}^{++}_{\mathcal{CQ}}}(\mathcal{O}(i)) = (\mathcal{T}(i), CT(i), QT(i), TG(i))$, where $TG(i) = (N(i), E(i))$, and we want to erase $Er(i) \subseteq \mathcal{O}(i)$ from the $\mathcal{O}(i)$, then we can construct an erased approximation $eA_{fn,\mathcal{EL}^{++}_{\mathcal{CQ}}}(\mathcal{O}(i), Er(i)) = (\mathcal{T}^E(i), CT^E(i), QT^E(i), TG^E(i))$, where $TG^E(i) = (N^E(i), E^E(i))$, as follows:

1. $N^E(i) = N(i) \setminus \{\alpha | \alpha$ is reachable in $TG(i)$ from elements of $Er(i)\}$.
2. $E^E(i) = E(i) \setminus \{(\alpha, \beta) | (\alpha, \beta) \in E(i)$ and $\alpha \notin N^E(i)\}$.
3. $\mathcal{T}^E(i) = \mathcal{T}(i) \setminus \{\alpha | \alpha \in \mathcal{T}(i)$ and $\alpha \notin N^E(i)\}$.
4. $CT^E(i) = CT(i) \setminus \{(a, b) | a$ and b are not concept names in $\mathcal{T}^E(i)\}$.
5. $QT^E(i) = QT(i) \setminus \{(a, r, n) | a^{r,n}$ is not a concept name in $\mathcal{T}^E(i)\}$.

As we can see, all the entailments (including the erased axioms themselves) reachable from the erased axioms are removed from the approximation and the traceability graph. The corresponding traceability edges are removed as well.

Consequently, the reasoning results should be erased:

1. for any role r in $\mathcal{T}(i)$ and not in $\mathcal{T}^E(i)$, erasing all role subsumption involving R. And removing $R(r)$.
2. earsing all role subsumptions $r \sqsubseteq_* s \in N(i)$ and $r \sqsubseteq_* s \notin N^E(i)$.
3. for any basic concept A in $\mathcal{T}(i)$ and not in $\mathcal{T}^E(i)$, removing $S(A)$.
4. for any basic concepts A, B such that $B \in S(A)$, if $A \sqsubseteq B \notin N^E(i)$, then $S(A) = S(A) \setminus \{B\}$.
5. for any role name r, basic concepts A and B such that $(A, B) \in R(r)$, if $A \sqsubseteq \exists r.B \notin N^E(i)$, then $R(r) = R(r) \setminus \{(A, B)\}$.

Similar as the DRed approach, such erasure may lose some information. Using our old example $\mathcal{O}(i) = \{\{a\} \equiv A, \{a\} \sqsubseteq A, a : A\}$, we only have $(\{a\} \equiv A, \{a\} \sqsubseteq A) \in E(i)$. If $\{a\} \equiv A \in Er(i)$ then we will have $\{a\} \sqsubseteq A \notin N^E(i)$. Compared to $tA_{fn,\mathcal{EL}^{++}_{CQ}}(\mathcal{O}(i) \setminus Er(i))$ this loses information, because $\{a\} \sqsubseteq A$ still remains in $\mathcal{O}(i) \setminus Er(i)$ or can still be approximated from $a : A$. As a consequence, any further entailments reachable from $\{a\} \sqsubseteq A$ are all missing. We will need to re-approximate and re-derive these entailments with the added axioms.

5.2 Adding Axioms into Ontology

In the adding step, we need to re-perform the approximation and reasoning. Such re-performing can be optimized with the erased approximation. More precisely, suppose the erased approximation $eA_{fn,\mathcal{EL}^{++}_{CQ}}(\mathcal{O}(i), Er(i)) = (\mathcal{T}^E(i), CT^E(i), QT^E(i), TG^E(i))$, where $TG^E(i) = (N^E(i), E^E(i))$, and we want to add $Ad(i)$ into the ontology ($Ad(i)$ can be $\emptyset$), then we can construct an added approximation $aA_{fn,\mathcal{EL}^{++}_{CQ}}(\mathcal{O}(i), Er(i), Ad(i)) = (\mathcal{T}^A(i), CT^A(i), QT^A(i), TG^A(i))$, where $TG^A(i) = (N^A(i), E^A(i))$, as follows:

1. Generating $\mathcal{T}^A(i), CT^A(i)$ and $QT^A(i)$ from $\mathcal{O}(i) \setminus Er(i) \cup Ad(i)$ with respect to Def.2. But in Step 1, instead of initializing $\mathcal{T}^A(i), CT^A(i)$ and $QT^A(i)$ with $\emptyset$, initializing them with $\mathcal{T}^E(i), CT^E(i)$ and $QT^E(i)$, respectively.
2. Generating $TG^A(i)$ from $(\mathcal{T}^A(i), CT^A(i), QT^A(i))$ with respect to Def.4. But in Step 2, instead of initializing $N^A(i)$ and $E^A(i)$ with $\emptyset$, initializing them with $N^E(i)$ and $E^E(i)$, respectively.

In the first step, we re-approximate the updated ontology by making use of the remaining approximation after erasure. In the second, we re-compute the traceability graph

by making use of the remaining traceability graph after erasure. Compared with a re-approximation from scratch $tA(\mathcal{O}(i+1)) = (\mathcal{T}(i+1), CT(i+1), QT(i+1), TG(i+1))$ we have the following properties: $\mathcal{T}^A(i) = \mathcal{T}(i+1), CT^A(i) = CT(i+1), QT^A(i+1) = QT(i+1)$ and $TG^A(i) = TG(i+1)$.

Thus, the reasoning results can be updated accordingly. Simply applying the traced completion rules again then we will get the results on $\mathcal{O}(i+1)$. Because we have already obtained partial results on the S sets, R sets and role subsumptions, such a completion procedure will be more efficient than computing all results from scratch. And the reasoning is still tractable and sound:

Theorem 2. *For any ontology $\mathcal{O}(i)$, erased axioms $Er(i) \subseteq \mathcal{O}(i)$ and added axioms $Ad(i)$, performing erasing as introduced in Sec.5.1 and adding as introduced in Sec.5.2 will terminate in polynomial time w.r.t. $|CN_{\mathcal{O}(i)\cup Ad(i)}|+|RN_{\mathcal{O}(i)\cup Ad(i)}|$ and the results are the same as reasoning on $\mathcal{O}(i) \setminus Er(i) \cup Ad(i)$.*

6 Evaluation

We implemented the proposed approach in our REL reasoner. In order to evaluate its performance, we tested it with large ABox stream reasoning. Due to the lack of stream reasoning benchmark, we generated our test data from existing benchmark ontologies and simulated streams. The ontology we used is the well-known Lehigh University Benchmark (LUBM) ontology [2]. It has a simple TBox but arbitrarily large ABox. In our evaluation, we generate one university with 15 departments. 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.

We do the following pre-processing to simulate an ontology stream:

1. We randomly partition the ABox into 15 sub-ABoxes of same size. We call them $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{14}$.
2. We re-merged the TBox $\mathcal{T}$ and sub-ABoxes $\mathcal{A}_0$ to $\mathcal{A}_4$ into $\mathcal{O}(0)$, as the initial ontology for stream reasoning.
3. Hence, we generate streams as follows: $\mathcal{O}(i+1) = \mathcal{O}(i) \setminus \mathcal{A}_i \cup \mathcal{A}_{i+5}$. In other words, in each update, we erase the $\mathcal{A}_i$ from $\mathcal{O}(i)$ and add $\mathcal{A}_{i+5}$ into it to create $\mathcal{O}(i+1)$. Thus, $\mathcal{O}(1) = \mathcal{T} \cup \mathcal{A}_1 \cup \mathcal{A}_2 \ldots \mathcal{A}_5, \ldots, \mathcal{O}(10) = \mathcal{T} \cup \mathcal{A}_{10} \cup \ldots \mathcal{A}_{14}$.

Now we create a stream with 10 updates. In each update, 20% of the ABox is changed.

For each stream ontology $\mathcal{O}(i)$, we perform ABox reasoning and query for the types of all individuals. This is because, in LUBM a certain type of an individual can have multiple sources, e.g. direct assertion, or through some domain/range axioms, or through subsumption closure of its other types. While the relation between two individuals are usually either directly asserted, or derived through role subsumption, i.e. without multiple sources.

We accomplish such a reasoning task with both the naive approach (re-do reasoning on $\mathcal{O}(i)$ from scratch) and the stream reasoning approach. We record the time of both

[2] `http://swat.cse.lehigh.edu/projects/lubm/`

approaches to evaluate the effectiveness of the stream reasoning approach. The results are summarized in the following table (Table 1).

Table 1. Evaluation Results ($\mathcal{O}(0)$ shows the time for the initial ontology. $Max(\mathcal{O}(i))$, $Min(\mathcal{O}(i))$ and $Ave(\mathcal{O}(i))$ show the maximal, minimal and average time for the updated ontologies, respectively. Time unit is second.)

Approach	$\mathcal{O}(0)$	$Max(\mathcal{O}(i))$	$Min(\mathcal{O}(i))$	$Ave(\mathcal{O}(i))$
Naive	19.223	13.897	10.667	11.5997
Stream Reasoning		3.982	2.175	2.5076

From the table we can see that, when the updating part makes up to 20% of the ABox, the stream reasoning approach is still significantly faster than the naive approach.

It is interesting to see how many percentages of the ABox can be updated while the stream reasoning approach remains faster than the naive approach. To answer this question, we further process the sub-ABoxes as follows:

1. We merged n (n=4,5,6,7) sub-ABoxes with the TBox to make the initial ontology, i.e. $\mathcal{O}(0) = \mathcal{T} \cup A_0 \cup \ldots A_{n-1}$.
2. For each n, the updated part varies from 1 sub-ABox to $(n-1)$ sub-ABoxes.

For example, when $n = 5$, we update the ontology with $1, 2, 3, 4$ sub-ABoxes, respectively. Thus, the *Update/Ontology ratio* (update ratio for short, ABox only) is $20\%, 40\%, 60\%$ and 80%, respectively.

For each n and each update ratio, we do same reasoning as before with both the naive approach and the stream reasoning approach, and then compute the $T_{Stream\ Reasoning}/T_{Naive}$ ratio (time ratio for short), where $T_{Stream\ Reasoning}$ and T_{Naive} are time for the two approaches, respectively. The relation between update ratio and time ratio is illustrated in Figure 1.

From this figure, we can see that stream reasoning approach always pays off no matter how large the ABox is, and how large the updated part is. This justifies the usability of our approach.

7 Further Extensions

Although not implemented, we introduce two extensions of our approach in this section. Both of them try to trade time consumption with space consumption:

1. **Maintaining Erased Information:** there is a possibility that erased axioms may be added back into the ontology again in the future. With the current solution, we still need to re-compute all information related to them. A potential extension is that, instead of immediately removing all corresponding information in erasure, we label them as "erased" and do not use them in subsequent approximation and reasoning. When they are added back into the ontology, we remove the "erased" label and certain results can be enabled immediately without reasoning. Consequently the

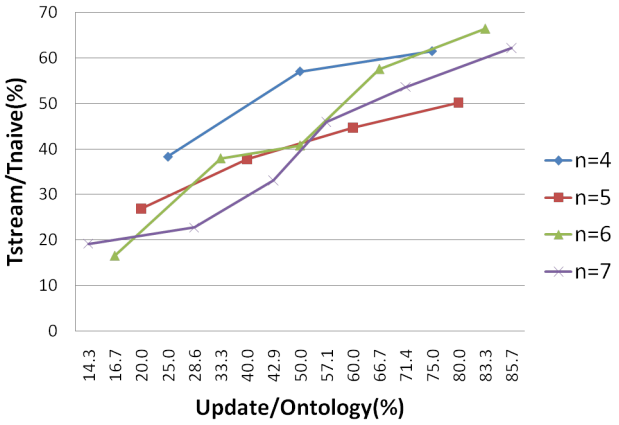

Fig. 1. The relation between update ratio and time ratio.

size of the TG will increase with the stream, i.e. $|TG|$ is polynomial w.r.t. $|\mathcal{O}(0) \cup Ad(1) \cup Ad(2) \cup \ldots|$.

2. **Maintaining Complete Traceability Information:** as our early example $\{\{a\} \equiv A, \{a\} \sqsubseteq A, a : A\}$ shows, an entailment $\{a\} \sqsubseteq A$ can have multiple sources. In the current solution we only maintain one of them in the TG. So that when such a source is erased, we need to re-perform approximation and reasoning to obtain the entailment. A potential extension is to maintain traceability information from all the sources in the TG and erase an entailment only if all the asserted axioms it is reachable from are erased. For example, when erasing $\{a\} \equiv A$ from the above ontology, $A \sqsubseteq \{a\}$ should be erased but $\{a\} \sqsubseteq A$ should not. Therefore entailments have multiple sources do not need to be re-computed. However, the complexity of this extension is comparable to computing all justifications, i.e. the computation is NP-Complete w.r.t. $|\mathcal{O}(i) \cup Ad(i)|$.

8 Conclusion

In this paper, we presented an approach to ontological stream reasoning via syntactic approximation. We took advantage of the complexity reduction brought by approximation technology and enabled stream reasoning for relatively complex languages and large knowledge bases by extending syntactic approximation with a traceability graph. And then we use such a graph to erase entailments from the ontology, and use incremental reasoning facility to handle the added axioms. Evaluation on benchmark ontology shows that our approach works nicely in practice. In the future, we would like to further improve the scalability of our approach. Combining with the time tag approach is also a potential improvement.

References

1. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ Envelope. In *Proceedings IJCAI-05*, 2005.
2. Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL Envelope Further. In Kendall Clark and Peter F. Patel-Schneider, editors, *In OWLED-2008*, 2008.
3. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
4. Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, and Michael Grossniklaus. An execution environment for c-sparql queries. In *EDBT '10*, 2010.
5. Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-sparql: Sparql for continuous querying. In *WWW2009*, 2009.
6. Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Incremental reasoning on streams and rich background knowledge. In *ESWC2010*, 2010.
7. Andre Bolles, Marco Grawunder, and Jonas Jacobi. Streaming sparql extending sparql to process data streams. In *ESWC08*, 2008.
8. Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *SIGMOD '93*, 1993.
9. Bernhard Hollunder, Werner Nutt, and Manfred Schmidt-Schauß. Subsumption Algorithms for Concept Description Languages. In *ECAI-90*, pages 348–353. Pitman Publishing, 1990.
10. Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The Even More Irresistible SROIQ. In *KR 2006*, 2006.
11. Sik Chun (joey) Lam, Jeff Z. Pan, Derek Sleeman, and Wamberto Vasconcelos. A fine-grained approach to resolving unsatisfiable ontologies. In *In Proc. of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI-2006)*, 2006.
12. Marko Luther and Sebastian Bohm. Situation-Aware Mobility: An Application for Stream Reasoning. In *in Proc. of 1st International Workshop on Stream Reasoning (SR2009)*, 2009.
13. Bijan Parsia, Christian Halaschek-wiener, and Evren Sirin. E.s.: Towards incremental reasoning through updates. In *in OWL DL. In: Proc. WWW-2006. (2006)*, 2006.
14. Yuan Ren, Jeff Z. Pan, and Yuting Zhao. Soundness Preserving Approximation for TBox Reasoning. In *the Proc. of the 25th AAAI Conference Conference (AAAI2010)*, 2010.
15. Yuan Ren, Jeff Z. Pan, and Yuting Zhao. Towards Soundness Preserving Approximation for ABox Reasoning of OWL2. In *the Proc. of the International Description Logic Workshop (DL2010)*, 2010.
16. Boontawee Suntisrivaraporn. Module Extraction and Incremental Classification: A Pragmatic Approach for $\mathcal{EL}^+$ Ontologies. In *ESWC'08*, 2008.
17. Emanuele Della Valle, Stefano Ceri, Daniele Braga, Irene Celino, Dieter Frensel, Frank van Harmelen, and Gulay Unel. Research Chapters in the Area of Stream Reasoning. In *in Proc. of 1st International Workshop on Stream Reasoning (SR2009)*, 2009.
18. Emanuele Della Valle, Stefano Ceri, Frank van Harmelen, and Dieter Fensel. It's a streaming world! reasoning upon rapidly changing information. *IEEE Intelligent Systems*, 24:83–89, 2009.
19. Raphael Volz, Steffen Staab, and Boris Motik. Incementally maintaining materializations of ontologies stored in logic databases. *In Journal of Data Semantics II, LNCS, Vol 3360*, 2:1–34, 2005.