WP6-D3

Stream Querying for Private Data

Project full title:	Knowledge Driven Data Exploitation
Project acronym:	K-Drive
Grant agreement no.:	286348
Project instrument:	EU FP7/Maria-Curie IAPP/PEOPLE WP 2011
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IBM/WP6-D3/D/PU/b1
Responsible editors:	Yuting Zhao
Reviewers:	Honghan Wu
Contributing participants:	IBM, UNIABDN
Contributing workpackages:	WP6
Contractual date of deliverable:	31 Oct 2015
Actual submission date:	31 Oct 2015

Abstract
Due to the dynamic nature of knowledge and data in semantic applications, *i.e.*, ontology stream querying technologies are essential for knowledge driven data exploitation systems. Nowadays, many proposed stream reasoning solutions and implemented systems apply forward chaining completion algorithms to handle the removal and addition of axioms. In this deliverable, we propose a novel approach to ontology stream querying that utilizes existing Database tools. Compared to existing works, this approach has a special advantage: the performance of query answering is automatically improved by the improvement of the state-of-the-art Database technologies.

Keyword List
ontology stream, stream query answering

Project funded by the European Commission within the 7th Framework Programme/Maria Curie Industry-Academia Partnerships and Pathways schema/PEOPLE Work Programme 2011.

Stream Querying for Private Data

Yuting Zhao[1], **Jeff Z. Pan**[2], **Jhonatan Garcia**[2] **and Alessandro Faraotti**[1]

[1] IBM Italy, Rome, Italy
Email: {yuting.zhao, alessandro.faraotti}@it.ibm.com
[2] Department of Computing Science, Aberdeen University, UK
Email: {jeff.z.pan, jgarcia}@abdn.ac.uk

31 Oct 2015

Abstract
Due to the dynamic nature of knowledge and data in semantic applications, *i.e.*, ontology stream querying technologies are essential for knowledge driven data exploitation systems. Nowadays, many proposed stream reasoning solutions and implemented systems apply forward chaining completion algorithms to handle the removal and addition of axioms. In this deliverable, we propose a novel approach to ontology stream querying that utilizes existing Database tools. Compared to existing works, this approach has a special advantage: the performance of query answering is automatically improved by the improvement of the state-of-the-art Database technologies.

Keyword List
ontology stream, stream query answering

Contents

1 Introduction

Recent years have witnessed the wide recognition of the importance of ontology and rule in the AI research and application.

As we know, ontologies are machine-readable compilations of human thoughts and serve as knowledge infrastructure in many different application domains to support knowledge management. In order to facilitate automatic processing of ontologies, today's *de facto* standard ontology languages, the Web Ontology Languages (OWLs), are based on a family of formal knowledge representations called the Description Logics [Baader et al., 2003] (DLs). Using DLs, ontologies can be considered as a set of logical axioms, describing the domain of discourse with different concepts, roles and individuals.

Rules in the Web have become another mainstream topic these days. Inference rules, such as SWRL rules [Horrocks et al., 2004], can be marked up for e-applications, such as e-commerce and e-science. The World Wide Web Consortium (W3C) has set up a Rule Interchange Format (RIF) Working Group[1] to develop a set of dialects, each of which can be used to exchange rules among a category of rule engines.

Ontologies and rules are closely related. In the one hand, some sub-profiles of ontology and rules can be transformed from one to the other. For example, it has been shown[2] that the RIF Core Dialect can express the inference rules for OWL 2 RL, which is a rules-oriented profile of the OWL 2 standard ontology language.

As we have outlined in Deliverable D6.1, in real world knowledge is usually subject to change. The dynamics in the different use cases in the K-Drive project has also been described in the the Deliverable D5.1. As D6.1 has introduced, the dynamics of ontologies have brought many new research challenges. In this deliverable, we are particularly interested in the development of *stream querying* technologies, which update query answering results affected by the updating of the ontology without naively re-computing all results.

In order to reuse previously computed results, many existing work on stream reasoning [Volz et al., 2005, Barbieri et al., 2010, Ren and Pan, 2011, Kazakov and Klinov, 2013] adopted the Delete and Re-derive (DRed) strategy [Gupta et al., 1993]. With DRed, a stream reasoner first over-estimates and over-deletes the reasoning results affected by the deleted original axioms, unaffected results are preserved. Then the reasoner re-derives the over-deleted results that can be inferred by the preserved axioms. In DRed, reducing the over-deletion and re-derivation is crucial to the performance of stream reasoning. In order to completely avoid the re-derivation in the DRed strategy, a counting approach [Urbani et al., 2013] has also been developed to pinpoint the affected results. Despite of the numerous differences among these approaches, as will be detailed later, all these approaches adopt forward chaining consequence-based algorithms to compute reasoning results.

In this deliverable, we analyse the pros and cons of these different approaches and present a novel ontology stream query answering ... approach with the DRed strategy by combining forward chaining and backward chaining of consequence-based algorithms. In comparison to existing works, such an approach has the following advantages:

1. It can be applied with either bookkeeping (such as using counting [Urbani et al., 2013] or using a truth maintenance system [Ren and Pan, 2011]) or non-bookkeeping (such as using context [Kazakov and Klinov, 2013]).

2. It can be parallelised by multiple computational cores with shared main memory, hence does not affect the concurrency of parallel reasoners [Kazakov and Klinov, 2013].

[1] http://www.w3.org/2005/rules/wiki/RIF_Working_Group

[2] http://www.w3.org/2005/rules/wiki/OWLRL

3. It helps to reduce the volume of over-deletion and re-derivation in the DRed strategy, as we will show both theoretically and empirically.

4. It can be applied on any knowledge representation whose reasoning can be realised with a consequence-based algorithm. When applied with a tractable algorithm, our approach is also tractable.

The remainder of the deliverable is organised as follows: In Sec. 2 we introduce the description logics, consequence-based reasoning and stream reasoning. In Sec. 3 we motivate our research by reviewing existing works on stream reasoning and systematically analysing their pros and cons. In Sec. 4 we present our approach and theoretically demonstrate its formal characteristics. Our proposal is implemented and empirically evaluated in Sec. 5. Section 6 summaries the deliverable and point out directions for future work.

This deliverable focuses on the development and evaluation of a novel stream reasoning technique. The technology described in this deliverable is part of the framework presented in D6.1. While this deliverable provides more concrete technical details. Such technology can be applied to perform stream reasoning over public semantic data streams, which are often encountered in case studies described in deliverables of WP1 and WP2. The types of data dynamics in such case studies are analysed in D5.1. Part of the contents of this deliverable, including the introduction of DLs, ontology stream and existing work on stream reasoning, has been introduced in D6.1. However, in order to make this deliverable self-contained, we will revisit such content in this deliverable with appropriate level of details.

2 Background

2.1 Description Logics

The Description Logics describe the domain of discourse with so called *concept* and *role expressions* and their *individuals*. A *signature* $\Sigma = (\mathcal{CN}, \mathcal{RN}, \mathcal{IN})$ consists of three mutually disjoint sets of concepts names $\mathcal{CN}$, roles names $\mathcal{RN}$ and individuals $\mathcal{IN}$. In DLs, *concept expressions* (or *concepts* for short) and *role expressions* (or *roles* for short) can be constructed inductively from a signature with different constructors. In this deliverable, we use the DL $\mathcal{EL}^+$ as an example to introduce DL.

$\mathcal{EL}^+$, the underpinning logic of OWL 2 EL [Motik et al., 2008], supports the following concept constructors:

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

in which $A \in \mathcal{CN}$, C and D are $\mathcal{EL}^+$ concepts and $R \in \mathcal{RN}$.

An $\mathcal{EL}^+$ ontology $\mathcal{O}$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. $\mathcal{T}$ is a finite set of general concept inclusions axioms (**GCI**s) of form $C \sqsubseteq D$, where C and D are both concepts, and role inclusion axioms (**RIA**s) of form $R_1 \sqsubseteq R_2$ or $R_1 \circ R_2 \sqsubseteq R_3$, where $R_{(i)} \in \mathcal{RN}$. $\mathcal{A}$ is a finite set of concept assertions ($a : C$), role assertions ($(a, b) : R$) and individual equality ($a \doteq b$) axioms.

An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty domain $\Delta^{\mathcal{I}}$ and a function $\cdot^{\mathcal{I}}$ that maps each atomic concept $A \in \mathcal{CN}$ to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each atomic role $R \in \mathcal{RN}$ 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 expressions. Particularly, given an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, concepts C, D and role R, the interpretations of complex concepts and roles are inductively defined as in Table 1.

An axiom α is satisfied by $\mathcal{I}$, written $\mathcal{I} \models \alpha$, *iff* (if and only if) the corresponding situation in Table 2 holds.

$\mathcal{I}$ is a model of $\mathcal{O}$, written $\mathcal{I} \models \mathcal{O}$, *iff* it satisfies all axioms of $\mathcal{O}$. An axiom α is an *entailment* by an ontology $\mathcal{O}$, written $\mathcal{O} \models \alpha$, *iff* all models of $\mathcal{O}$ satisfies α. In the remainder of this deliverable, we will

Table 1: $\mathcal{EL}^+$ Complex Expression Interpretation

Expression E	Interpretation E^I
$\top$	$\Delta^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$\exists R.C$	$\{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}}, y \in C^{\mathcal{I}}\}$

Table 2: $\mathcal{EL}^+$ Axiom Satisfiability

Axiom α	$\mathcal{I} \models \alpha$ *iff*
$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
$R_1 \circ R_2 \sqsubseteq R_3$	$R_1^{\mathcal{I}} \times \cdots \times R_n^{\mathcal{I}} \subseteq R_3^{\mathcal{I}}$
$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
$a : C$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
$(a, b) : R$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$
$a \doteq b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$

use *axiom* and *entailment* interchangeably when it is clear from context that the axiom is an entailment of some ontology. 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 $J' \subset J_{\mathcal{O}}(\alpha)$.

2.2 Consequence-based Reasoning

Consequence-based algorithms are a family of DL reasoning algorithms that gain popularity in the recent years due to their capability of simultaneously deriving a set of entailments without constructing models. A consequence-based algorithm usually consists of two closely related components, one is a set of completion rules, the other is a serialised forward-chaining procedure to apply the rules.

As an example, below is a set of $\mathcal{EL}^+$ TBox completion rules, in which $\sqsubseteq^*$ is the transitive, reflexive closure of $\sqsubseteq$ in $\mathcal{O}$. We call this rule set the **R** rules. Each rule has 2 or 3 parts: the axioms above the bar are the premises; the axioms below the bar are the consequences; and the part right to the bar are the side conditions. **R** rules are similar to the rules presented by Kazakov and Klinov [Kazakov and Klinov, 2013] with a few differences we will explain later.

$$\mathbf{R}_{\sqsubseteq}\frac{C \sqsubseteq D, D \sqsubseteq E}{C \sqsubseteq E}$$

$$\mathbf{R}^{-}_{\sqcap}\frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1, C \sqsubseteq D_2}$$

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

$$\mathbf{R}_{\exists}\frac{E \sqsubseteq \exists R.C, C \sqsubseteq D, R \sqsubseteq^* S}{E \sqsubseteq \exists S.D} : \exists S.D \text{ occurs in } \mathcal{O}$$

$$\mathbf{R}_{\circ}\frac{E \sqsubseteq \exists R_1.C, C \sqsubseteq \exists R_2.D, R_1 \sqsubseteq^* S_1, R_2 \sqsubseteq^* S_2, S_1 \circ S_2 \sqsubseteq S}{E \sqsubseteq \exists S.D}$$

Given an ontology, a consequence-based algorithm repeatedly applies the completion rules until

no more rule can be applied. The results of such a procedure can be characterised by the following definition:

Definition 1 ***(Completion Closure)*** *For a set of axioms S and a completion rule set R, the immediate results of applying R on S, denoted by $R(S)$ or $R^1(S)$, is the set of axioms that are either in S, or can be derived as consequence from premises in S by a single rule in R.*

Let $R^{n+1}(S) = R(R^n(S))$ for $n \geq 1$, then the completion closure *of S w.r.t. R, denoted by $R^*(\mathcal{O})$, is some $R^n(S)$ s.t. $R^n(S) = R(R^n(S))$.*

A rule set R converges *if for any $\mathcal{O}$, $R^*(\mathcal{O})$ exists.*

It can be show that the following properties holds:

Lemma 1 *Let $S_{(i)}$ be sets of axioms, R a set of completion rules and $n \geq 1$:*

$$S_1 \subseteq S_2 \rightarrow R^n(S_1) \subseteq R^n(S_2) \tag{1}$$

$$R^*(R^n(S)) = R^*(S) \tag{2}$$

$$R^*(S_1 \cup S_2) = R^*(R^n(S_1) \cup S_2) \tag{3}$$

Proof Property (1): For any $\alpha \in R(S_1)$, α is either in S_1, or can be derived as a consequence from premises in S_1. In the former case, α is also in S_1. In the latter case, α can be derived from the same premises in S_2. Hence $R(S_1) \subseteq R(S_2)$. Inductively, $R^2(S_1) = R(R(S_1)) \subseteq R(R(S_2)) = R^2(S_2)$. Similar we have $R^n(S_1) \subseteq R^n(S_2)$ for any n.

Property (2): On the one hand, according to the definition, we have $R^n(S) \subseteq R^{n+1}(S) \subseteq \cdots \subseteq R^*(S)$. Hence $R^*(R^n(S)) \subseteq R^*(R^*(S)) = R^*(S)$. On the other hand, since $S \subseteq R^n(S)$, we have $R^*(S) \subseteq R^*(R^n(S))$. Together we have $R^*(R^n(S)) = R^*(S)$.

Property (3): The $\rightarrow$ direction can be proved in a similar way as above. On the $\leftarrow$ direction, we have $R^n(S_1) \subseteq R^*(S_1 \cup S_2)$ and $S_2 \subseteq R^*(S_1 \cup S_2)$. Hence we have $R^n(S_1) \cup S_2 \subseteq R^*(S_1 \cup S_2)$. Similar as before, we have $R^*(R^n(S_1) \cup S_2) \subseteq R^*(R^*(S_1 \cup S_2)) = R^*(S_1 \cup S_2)$. Together we have $R^*(S_1 \cup S_2) = R^*(R^n(S_1) \cup S_2)$. □

In this deliverable, we make the following assumptions regarding a completion rule set for DL $\mathcal{L}$:

1. The premises and consequences of each completion rule are all axioms in $\mathcal{L}$;
2. Each completion rule is minimal, i.e., there is no other rule with less premises or side conditions but deriving the same consequence;
3. Each rule is soundness-guaranteed, i.e. the consequences are entailed by premises and side conditions;
4. The rule set converges;

The first assumption ensures that the consequence-based algorithm operates on a set of entailments in $\mathcal{L}$. The second assumption ensures that the reasoning procedure does not require unnecessary premises. The third assumption ensures that the reasoning results are sound. The last assumption ensures that the completion closure exists. To the best of our knowledge, these four assumptions are satisfied by all the consequence-based algorithms developed so far, including the reasoning techniques developed in the K-Drive project.

In a consequence-based algorithm, the completion closure can be computed in a forward-chaining manner. Starting from the initial S, it always checks which rule can be applied on which existing

axioms $\alpha_1, \ldots, \alpha_n$ in S to derive a new axiom $\beta \notin S$ and then adds β into S. We call such a step of deriving β from $\alpha_1, \ldots, \alpha_n$ an *execution* of the rule. S is then repeatedly expanded until it converges to $R^*(S)$. Such a procedure can be conceptually described with the help of a list L as in the following algorithm *FCC (Forward Chaining Completion)*.

Forward Chaining Completion:
$FCC(L, S, R)$
INPUT: a list of axioms to be processed L, a set of processed axioms S, a completion rule set R
OUTPUT: a set of processed axioms S

1: **while** $L \neq \emptyset$ **do**
2: get an element $\alpha \in L$
3: $L := L \setminus \{\alpha\}$, $S := S \cup \{\alpha\}$
4: **for** each $rule$ in R **do**
5: **if** α can be used as a premise, and all other premises $\alpha_2, \ldots, \alpha_n$ are in S, and consequence β is not in $S \cup L$ **then**
6: execute $rule$ and add β into L
7: **return** S

This algorithm processes axioms in L one after another. When an axiom α is processed, it checks if it can be used to execute a rule with other axioms in S to infer $\beta \notin S \cup L$, which implies that β has not be processed or derived by other executions yet. If that is the case, the rule will be executed and β will be added into L. In any case, processed axiom α will be moved from L to S. L can either be a first-in-first-out queue, or a first-in-last-out stack, or a mixture of both. For example, the ELK reasoner [Kazakov et al., 2013] implements L as a queue. While the CEL reasoner [Baader et al., 2005] implements L as a mixture of queue and stack.

It can be shown that $R^*(L) = FCC(L, \emptyset, R)$:

Lemma 2 *If $R(S) \subseteq S \cup L$, then $R^*(S \cup L) = FCC(L, S, R)$.*

Proof Let us use $L_0, L_1, \ldots, L_n$ and $S_0, S_1, \ldots, S_n$ to denote the different Ls and Ss when Step-1 of the algorithm is checked, respectively, we can prove the $\rightarrow$ direction of this lemma with induction.

First of all, we know that $S_n = FCC(L, S, R)$.

We then show that $R(S_i) \subseteq S_i \cup L_i$ implies $R(S_{i+1}) \subseteq S_{i+1} \cup L_{i+1}$. This is because:

1. According to Step-2 and 3 of the algorithm, $S_{i+1} = S_i \cup \{\alpha\}$, where α is an element of L_i.
2. Let $B = \{\beta | \beta$ is a new consequence with α and axioms in S_i as premises$\}$, apparently, $R(S_{i+1}) = R(S_i) \cup B$.
3. According to Step-2, 5 and 6 of the algorithm, we have $L_{i+1} = L_i \setminus \{\alpha\} \cup B$.
4. Combined with $R(S_i) \subseteq S_i \cup L_i$, we have $R(S_{i+1}) = R(S_i) \cup B \subseteq S_i \cup L_i \cup B = (S_i \cup \{\alpha\}) \cup (L_i \cup B \setminus \{\alpha\}) = S_{i+1} \cup L_{i+1}$.

Since $L_0 = L$ and $S_0 = S$, when $R(S) \subseteq S \cup L$, we have $R(S_1) \subseteq S_1 \cup L_1$,.... Eventually, when S and L no longer change, i.e. $L_n = \emptyset$, we have $R(S_n) \subseteq S_n \cup L_n = S_n = FCC(L, S, R)$. According to the algorithm, we have $S \cup L \subseteq S_n$. According to (1) in Lemma 1, we have $R^*(S \cup L) \subseteq R^*(S_n) \subseteq S_n = FCC(Q, S, R)$.

The $\leftarrow$ direction of the lemma is trivial. Apparently, every axiom in $FCC(L, S, R)$ are directly or indirectly derived from axioms in $L \cup S$. It is obvious that $FCC(L, S, R) \subseteq R^*(S \cup L)$.

Hence the lemma is proved. □

Since $R^*(\emptyset) \subseteq L \cup \emptyset$, we have $R^*(L) \subseteq FCC(L, \emptyset, R)$.

With the above procedure, consequence-based algorithms can be used to perform ontology reasoning. For example, with the **R** rules, classification (computing atomic concept hierarchies) of an $\mathcal{EL}^+$ ontology $\mathcal{O}$ can be realised by executing $FCC(L, \emptyset, \mathbf{R})$, where $L = \mathcal{O} \cup \{C \sqsubseteq C | C \text{ occurs in } \mathcal{O}\} \cup \{C \sqsubseteq \top | C, \top \text{ occur in } \mathcal{O}\}$. Here we realise the $\mathbf{R}_0$ and $\mathbf{R}_\top$ by Kazakov and Klinov [Kazakov and Klinov, 2013] with the initialisation of L. It can be shown that such results are complete for classification, i.e. for $A, B \in \mathcal{CN}_\mathcal{O}$, $\mathcal{O} \models A \sqsubseteq B$ *iff* $A \sqsubseteq B \in FCC(L, \emptyset, \mathbf{R})$. ABox reasoning can be achieved in a similar manner by rewriting ABox axioms into TBox axioms and treating individuals as atomic concepts. Such treatment allows us to deal with large scale datasets in the K-Drive use cases.

Kazakov et al. [Kazakov et al., 2013] showed that the completion rules **R** can be implemented to support lock-free parallel reasoning. Particularly, as one shall notice, for each rule, the RIA premises (if there is any) can be precomputed and used as side conditions. For example $\mathbf{R}_\exists$ can be rewritten as follows:

$$\frac{E \sqsubseteq \exists r.C, C \sqsubseteq D}{E \sqsubseteq \exists s.D} : r \sqsubseteq^* s, \exists s.D \text{ occurs in } \mathcal{O}$$

The remaining GCI premises share a common concept component, such as the C in the above example. Such a concept is called the *context* of the corresponding premise axioms. Given such context-based rules, when processing an axiom α as in Step-5 of algorithm FCC, it is certain that all other premises will have a same context C as α. Hence the search of possible rule executions can be performed exclusively on a subset of S that have C as a context. To this end, the algorithm can partition the L and S into sub-lists $L(C_1), \ldots, L(C_n)$ and sub-sets $S(C_1), \ldots, S(C_n)$ w.r.t. the contexts of axioms, i.e. $L(C_i)$ and $S(C_i)$ contain only axioms with C_i as a context. And the processing of different $L(C_i)$s can be performed in parallel, since they will access different $S(C_i)$s, respectively. When the algorithm is executed by multiple computational cores, each context is processed by at most one core at any time. This ensures that each core will never have to wait for another core to finish its job.

Consequence-based algorithms have been widely used and rapidly developed. Besides the $\mathcal{EL}$ family of DLs, consequence-based algorithms have been developed for OWL 2 RL [Krötzsch, 2012], Horn-$\mathcal{SHIQ}$ [Kazakov, 2009], $\mathcal{ALCH}$ [Simančík et al., 2011]. The reasoning of RDF(S), which are defined by rules of corresponding entailment regimes, can also be realised by consequence-based algorithms. Hence, the approach we are going to present in this deliverable will be applicable to all the above knowledge representations.

2.3 Ontology Stream Reasoning

D6.1 presented a definition of ontology stream proposed in our previous work [Ren and Pan, 2011], which generalised the idea of ontology versioning [Barbieri et al., 2010] and linear version space [Huang and Stuckenschmidt, 2 to define the notion of an ontology stream as a sequence of ontologies $\mathcal{O}_0, \ldots, \mathcal{O}_n$, in which each $\mathcal{O}_i$ is called a snapshot. When the ontology is updated from a snapshot $\mathcal{O}_i$ to $\mathcal{O}_{i+1}$, the *addition* is defined as $\mathcal{O}_{i+1} \setminus \mathcal{O}_i$ and the *deletion* is defined as $\mathcal{O}_i \setminus \mathcal{O}_{i+1}$.

In ontology streams, a typical reasoning challenge is to update the reasoning results when the ontology is updated. Existing works have investigated this problem on the computation of completion closure. Particularly, when L is updated to L', $FCC(L, \emptyset, R)$ should also be updated to $FCC(L', \emptyset, R)$. There

are two different approaches to solve this problem. One is *Naive Reasoning*, which recomputes all results completely. The other is *Stream Reasoning*, which attempts to re-use the results of $FCC(L, \emptyset, R)$ to compute $FCC(L', \emptyset, R)$, without completely re-computing $FCC(L, \emptyset, R)$. The later is the focus of this deliverable.

Such a stream reasoning problem covers the different types of reasoning services (snapshot/windowed services) and ontology stream dynamic forms introduced in the D6.1. Particularly, to support snapshot service, the particular snapshot can be treated as the updating ontology; To support windowed service, the collection of all snapshots in the window can be treated as the updating ontology. When the window moves forward, it can be regarded as the ontology is updated with the removal of the earliest snapshot and the addition of the next snapshot. Also, to support regular dynamics, one only needs to synchornise the deletion and addition, while random dynamics can be naturally supported. Monotonic dynamics can be supported with a special kind of stream reasoning, in which the deletion is always empty. Such stream reasoning is incremental reasoning, as we will introduce in the next section.

3 Technical Motivation

3.1 State Of The Art

Ontology streams bring many new research challenges. Apart from the design of new ontology representation and querying languages [Bolles et al., 2008, Barbieri et al., 2009], the prediction of future streams [Lecue and Pan., 2013] and explanation of past streams [Klarman and Meyer, 2013], the belief revision of knowledge in ontologies [Qi and Yang, 2008] and the maintenance of stream with limited space [Gao et al., 2014], many research efforts have been directed to topics relevant to stream reasoning.

Incremental reasoning can be considered as a special case of stream reasoning in which the deletion is always empty: let $\mathcal{O}$ be an ontology and $S = FCC(\mathcal{O}, \emptyset, R)$ be its reasoning results w.r.t. rule set R, let Add be a set of new axioms added into $\mathcal{O}$, it is obvious that we have $S \subseteq FCC(\mathcal{O} \cup Add, \emptyset, R)$. Incremental reasoning attempts to compute only $FCC(\mathcal{O} \cup Add, \emptyset, R) \setminus S$ but not to completely recompute S.

Cuenca Grau et al. [Cuenca Grau et al., 2010] have developed a generic black-box approach to incremental reasoning for DLs as expressive as $\mathcal{SROIQ}$. This approach is known to have high complexity. Its specialisation on $\mathcal{EL}^+$ coincides with the reachability-module-based duo-ontology classification method for $\mathcal{EL}^+$ [Suntisrivaraporn, 2008]. Interestingly, this $\mathcal{EL}^+$ incremental reasoning method can be generalised to other consequence-based algorithms by reusing Algorithm *FCC*. In fact, let $S = FCC(\mathcal{O}, \emptyset, R)$, it can be shown that $FCC(\mathcal{O} \cup Add, \emptyset, R) = FCC(Add, S, R)$: $FCC(\mathcal{O} \cup Add, \emptyset, R) = R^*(\mathcal{O} \cup Add)$. Since $S = R^*(\mathcal{O})$ is a closure, we have $S \subseteq S \cup Add$. According to Lemma 2, we have $R^*(S \cup Add) = FCC(Add, S, R)$. Therefore $FCC(\mathcal{O} \cup Add, \emptyset, R) = FCC(Add, S, R)$.

Comparing the execution of complete re-computation $FCC(\mathcal{O} \cup Add, \emptyset, R)$ and incremental reasoning $FCC(Add, S, R)$, we can see that $S = FCC(\mathcal{O}, \emptyset, R) = R^*(\mathcal{O})$ can be reused and does not need to be re-computed. Such a procedure for incremental reasoning is adopted by many stream reasoning solutions, including the ones we will introduce later.

Stream reasoning considers the dynamics of knowledge. There have been many works regarding **querying streaming data** on the semantic web. [Bolles et al., 2008] proposed an extension of SPARQL, the standard ontology querying language, to process data streams. Similarly, C-SPARQL [Barbieri et al., 2009], another extension of SPARQL can continuously query from an RDF knowledge base. Such query languages usually register a query to get the latest results from a data stream with two parameters: one

parameter is the *update interval*, which indicates the time between two consecutive updates of query results; the other is the *window size*, which indicates how much of the previous data should be considered in query answering. In other word, data added into the data stream older than the window size should be regarded as removed.

Comparing to handling the addition of new axioms, the deletion of existing axioms is more complex. In order to address this problem, Volz et al. [Volz et al., 2005] first adopted the Delete and Rederive (DRed) strategy [Gupta et al., 1993] from traditional data stream management systems and proposed a declarative variant of it. Such a strategy can be conceptually described with the following 3 stages:

1. **Over-deletion:** it first over-estimates the consequences of the deletion and then deletes all these consequences. Results that are not consequences of the deleted axioms will be preserved.
2. **Re-derivation:** it then re-derives the over-deleted consequences that can be derived by the preserved results.
3. **Incremental reasoning:** it finally adds new entailments that are derived from the new facts.

The incremental reasoning stage can be realised with the same mechanism we just introduced. Such a mechanism has also been adopted by all the existing stream reasoning approaches. Hence, in this deliverable we will focus on the optimisation of the over-deletion and re-derivation.

Let $\mathcal{O}$ be an ontology, $R^*(\mathcal{O}) = FCC(\mathcal{O}, \emptyset, R)$ be its completion closure w.r.t. R, $Del \subseteq \mathcal{O}$ be a set of axioms to remove. A DRed approach first identifies a set of valid over-deleted axioms $OD \subseteq R^*(\mathcal{O})$ w.r.t. Del, as defined in the following definition:

Definition 2 ***(Valid Over-deletion)*** *Let $\mathcal{O}$ be an ontology, $Del \subseteq \mathcal{O}$ a set of axioms to remove, R a completion rule set, an valid over-deletion OD of $R^*(\mathcal{O})$ w.r.t. Del satisfies the following conditions:*

1. *$\forall \alpha \in R^*(\mathcal{O})$, if for every $\mathcal{J}_{\mathcal{O}}(\alpha)$ it is true that $\mathcal{J}_{\mathcal{O}}(\alpha) \cap Del \neq \emptyset$, then $\alpha \in OD$.*
2. *$\forall \alpha \in R^*(\mathcal{O} \setminus Del)$, there is some $J = \mathcal{J}_{\mathcal{O}}(\alpha)$ such that $J \cap OD = \emptyset$.*

The first condition ensures that OD over-deletes all entailments that can only be derived from some axioms in Del. The second condition ensures that any entailment that can be entailed by $O \setminus Del$ will be entailed by $R^*(\mathcal{O}) \setminus OD$.

A DRed approach then re-derives any axiom $\alpha \in OD$ if there is some $\mathcal{J}_{\mathcal{O}}(\alpha)$ s.t. $\mathcal{J}_{\mathcal{O}}(\alpha) \cap Del = \emptyset$. Different DRed or non-DRed stream reasoning approaches differ primarily on how they identify the over-deleted results and how they perform re-derivation. We categorise them w.r.t. their re-derivation mechanism as follows:

1. *Global Re-derivation:* Volz et al. [Volz et al., 2005] rewrite and maintain ontologies in logical databases and updates them with change maintenance programs. The over-deletion and re-derivation are realised by executing the change maintenance programs. Francesco Barbieri et al. [Barbieri et al., 2010] made an additional assumption that the addition and deletion of data in a stream is synchronised and the time in between is known to the stream reasoner. With this assumption, the time when a result should be over-deleted can be pre-determined when it is derived. In our previous work [Ren and Pan, 2011], we developed a DRed stream reasoner with a Truth Maintenance System (TMS). A TMS is a loopless directed graph in which nodes denote the axioms in the completion closure, and edges connect premise/side condition axioms to consequence axioms. When some original axioms are deleted, all axioms to which the deleted axioms have paths in the TMS will be over-deleted. Consider the following example:

Example 1 *Figure 1 shows a TMS in over-deletion. In this figure,* $\mathcal{O} = \{\alpha_1, \alpha_2, \beta_1\}$ *and* $R^*(\mathcal{O}) = \mathcal{O} \cup \{\alpha_3, \alpha_4, \beta_2\}$*. When* $Del = \{\alpha_1\}$ *is deleted,* $OD = Del \cup \{\alpha_3\}$ *since according to the TMS,* α_3 *the only entailment connected from* α_1 *in the TMS.*

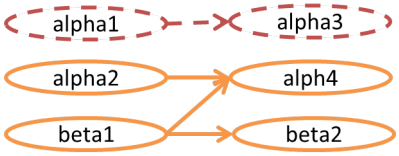

Figure 1: Over-deletion with a TMS

When performing re-derivation, all these approaches to some extent reuse the forward chaining procedure $FCC(R^*(\mathcal{O}) \setminus OD, \emptyset, R)$. It is apparent that this procedure satisfies the condition in Lemma 2 hence the re-derivation results are complete. Such global re-derivation DRed approaches have two major limitations:

(a) All these approaches require bookkeeping of either some maintenance program, or some temporal information, or a TMS to identify the over-deleted results OD. Such bookkeepings will impose performance and resource-consumption overhead to compute, maintain and update.

(b) The re-derivation has to go through all remaining axioms $R^*(\mathcal{O}) \setminus OD$ to ensure the completeness of results, even if many of them can not infer further entailments.

2. *Local Re-derivation:*A recent work by Kazakov and Klinov [Kazakov and Klinov, 2013] proposed a non-bookkeeping DRed approach to address the limitations mentioned above. It is applicable for consequence-based algorithms with contexts. The key point is to exploit the independent nature of different contexts to facilitate the over-deletion and re-derivation. Using $\mathcal{EL}^+$ with the **R** rules introduced in Sec. 2 as an example, given an ontology $\mathcal{O}$ and a set of axioms Del to delete, this approach first re-runs a similar forward chaining procedure as in algorithm *FCC* to identify a set of entailments $DEL = \{C \sqsubseteq D | C \sqsubseteq D$ can be directly or indirectly derived from premises in $Del\}$. Technically, this can be achieved by using Del as the initial list, and $R^*(\mathcal{O})$ as the initial set, of a forward chaining procedure and collect all the derived entailments into DEL.

   It then computes $Broken = \{C \sqsubseteq E | C \sqsubseteq E \in R^*(\mathcal{O}), C \sqsubseteq D \in DEL\}$. i.e. all derived GCIs whose LHS (left hand side) context occurring in some GCI that can be derived from the removed axioms. Below is an example:

Example 2 *Figure 2 shows a closure similar as the one in Figure 1. Now the closure is partitioned into two contexts.* α_i*s all have context* C_1 *and* β_i*s all have context* C_2*. Suppose we still have* $Del = \{\alpha_1\}$ *and* $DEL = Del \cup \{\alpha_3\}$*, since* α_2 *and* α_4 *also belongs to the same context, we have* $\{\alpha_2, \alpha_4\} \subseteq Broken$*.*

As we mentioned in Sec. 2, in such a parallel reasoning procedure, entailments of each context can be computed independently. Therefore one only needs to re-derive axioms in $Broken$, which means that entailments whose LHS does not occur in DEL as a LHS will be perserved. To re-derive $Broken$, the approach consider the following two types of over-deleted entailments:

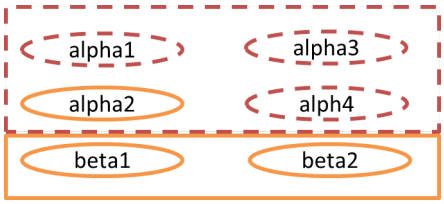

Figure 2: Over-deletion with Context

(a) The *Type-1* are entailments in $Broken \cap \mathbf{R}(\mathbf{R}^*(\mathcal{O}) \setminus Broken)$, i.e. entailments in $Broken$ that do not need to be derived from premises in $Broken$. Note that with the $\mathbf{R}$ rules, all consequences have the same LHS context as some premise. Hence, if a consequence is in $Broken$, it can not be re-derived from entailments not in $Broken$. This conveniently means that the *Type-1* are essentially the original axioms in $\mathcal{O} \setminus Del$ with some context C s.t. $C \sqsubseteq E \in Broken$. For example, in Figure 2, we have α_2 as a preserved original axiom, is of *Type-1*.

(b) The *Type-2* are entailments in $Broken$ that have to be derived from at least one premise in $Broken$. Since all such entailments are directly or indirectly derived from the *Type-1* entailments, one only needs to run a forward chaining procedure similar as FCC from *Type-1* axioms. Since *Type-1* axioms can be partitioned into different contexts, *Type-2* can be performed in parallel for each context.

Compared to the global re-derivation approaches, this approach avoids bookkeeping completely, and the re-derivation does not need to process all preserved axioms. Instead, it processes and re-derives only the axioms with the contexts affected by deletion. Similar as the original forward chaining procedure, over-deletion and re-derivation can also be parallelised w.r.t. each context.

Note that, when deriving *type-2* entailments from *type-1* entailments, namely the undeleted original axioms, this approach will essentially attempt to derive all entailments in $Broken \setminus \{\alpha | \alpha$ is *type-1*$\}$. In other words, all entailments in affected contexts are effectively over-deleted and will be re-derived from the remaining original axioms of those contexts. Hence, the effectively over-deleted axioms are $OD = Broken \setminus (\mathcal{O} \setminus Del)$.

3. *No Re-derivation:* In order to completely avoid the re-derivation in DRed, Urbani et al. [Urbani et al., 2013] applies the counting strategy proposed by Gupta et al. [Gupta et al., 1993] to precisely pinpoint the axioms that have to be removed from $R^*(\mathcal{O})$. It consists of the following important steps:

   (a) It uses a structure similar as a TMS to maintain the derivation relations from premises to consequences of rules.

   (b) For each entailment $\alpha \in R^*(\mathcal{O})$, it also uses a number $N(\alpha)$ to maintain how many independent rule executions can be used to derive α. If $\alpha \in \mathcal{O}$, $N(\alpha)$ is initialised with 1. Otherwise, $N(\alpha)$ is initialised with 0.

   (c) When doing reasoning, it uses a similar procedure as *FCC*, the major difference is that, in Step-5 of the algorithm, if a consequence β is already in $S \cup L$, it does not ignore the rule execution. Instead, it executes the rule and connect the premises and the β in the TMS, and then increase $N(\beta)$ by 1.

(d) When performing over-deletion, for each $\alpha \in Del$, it decreases $N(\alpha)$ by 1. If $N(\alpha) = 0$, it finds all β that are derived from α, and then decreases $N(\beta)$ by 1. If $N(\beta) = 0$, it propagates.

(e) Eventually, $\{\alpha \in R^*(\mathcal{O}) | N(\alpha) = 0\}$ are deleted.

This procedure can be illustrated with the following example:

Example 3 *The upper part of Figure 3 shows a TMS similar as the one in Figure 1. The main difference is that now the TMS recognised that α_3 can not only be derived from α_1, but also α_2. Hence its counting $N(\alpha_3) = 2$.*

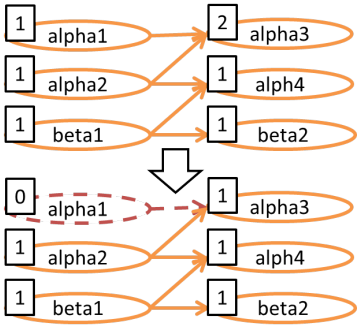

Figure 3: Over-deletion with Counting

The lower part of Figure 3 shows how the deletion works. When $Del = \{\alpha_1\}$, $N(\alpha_1) = 0$. Consequently $N(\alpha_3) = 1$. Since $N(\alpha_3) \neq 0$, α_3 will not be deleted and its derivation from α_2 is still preserved.

It was argued that the $\{\alpha \in R^*(\mathcal{O}) | N(\alpha) = 0\}$ produced as above is precisely the set of axioms that can not be derived by axioms in $\mathcal{O} \setminus Del$ hence re-derivation will not be needed, as illustrated by the example. Due to this advantage, such a counting approach is conceptually and empirically more efficient than DRed when dealing with removal of axioms. Parallelisation can also be supported.

3.2 Limitation of Existing Works

Despite of the progress of stream reasoning technologies and the attempt to avoid bookkeeping or re-derivation, existing stream reasoning solutions still have their limitations.

1. *Global Re-derivation:* As mentioned before, such approaches all require bookkeeping to identify OD efficiently and precisely. Another more important limitation of global re-derivation is that it re-derives axioms by re-processing all the preserved axioms in $R^*(\mathcal{O}) \setminus OD$. Even if some preserved axioms do not contribute to the re-derivation, it will still be re-processed. This is not "streaming" enough because axioms unaffected by or not contributing to the updating are still involved. These two limitations are exactly what the local re-derivation and no re-derivation approaches are trying to solve.

2. *Local Re-derivation:* This approach completely avoids bookkeeping to improve efficiency and memory-efficiency. But it also has limitations:

   (a) It almost always over-deletes more axioms than necessary. Taking the TMS-based global re-derivation as an example, let OD_{TMS} be the over-deleted axioms in global re-derivation, OD_L be the over-deleted axioms in local re-derivation, it can be shown that $OD_{TMS} \subseteq DEL \subseteq OD_L$:

      i. If an axiom was derived from $O \setminus Del$ and can also be derived from some axioms in Del, it will not be included in OD_{TMS} but it will certainly be included in DEL.

      ii. While the OD_L is almost always larger than DEL by definition.

   This implies that in the local re-derivation approach, some of the over-deletion are guaranteed to be unnecessary. In fact, DEL already satisfies the conditions for a valid over-deletion (Def. 2) and all entailments in $OD_L \setminus DEL$ are unnecessarily over-deleted and re-derived. Especially, if the Del includes the majority of all contexts, then the OD_L will include the majority of $R^*(\mathcal{O})$, even if for each context, only 1 entailment is actually affected by the removal of Del.

   (b) It is not applicable to consequence-based algorithms that do not have context. In this case, the computation of $R^*(\mathcal{O})$ cannot be parallelised. Consequently, the re-derivation can not be performed independently for each context. It is not clear if there is always a context-based variant to any consequence-based algorithm.

   (c) It is difficult to deal with non-premise axioms in Del. Note that DEL is computed by processing axioms in Del as premises. If Del includes some non-premise axiom α, α will not trigger any rule execution, hence the impact of its deletion on $R^*(\mathcal{O})$ will not be captured in DEL. Such non-premise axioms include axioms on side conditions, such as the axiom $r \sqsubseteq^* s$, as shown in the rewritten rule $\mathbf{R}_\exists$ in Sec. 2. This is part of the reason that local re-derivation [Kazakov and Klinov, 2013] does not support removal of RIAs. In fact, RIAs are not the only kind of side condition axioms. GCIs can also be used as side conditions in consequence-based algorithms. One can of course include side condition axioms as premises as well, just like in the original $\mathbf{R}_\exists$. However, doing so results in a completion rule whose premise axioms no longer have a shared component, hence the rule set can not be executed in parallel. Actually, replacing premise axioms with side conditions is an effective way to improve the efficiency of context-based algorithm. For example the rule $\mathbf{R}_\sqsubseteq$ can be rewritten into the following rule, as optimised in the ELK reasoner [Kazakov et al., 2013]:

$$\frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in \mathcal{O}$$

   However, with the above formulation, when the side condition $D \sqsubseteq E \in \mathcal{O}$ is deleted, it is difficult to obtain consequences such as the above $C \sqsubseteq E$ in DEL. Such a problem can be addressed by maintaining all the derived sub-concepts of D in $R^*(\mathcal{O})$. Nevertheless, such maintenance can be considered as some kind of bookkeeping.

3. *No Re-derivation:* The motivation of this approach is to completely aovid re-derivation in DRed by precisely deleting only the axioms that can not be re-derived. However, this approach has several issues:

   (a) Its results are not necessarily sound w.r.t. naive re-computation. Below is an example:

Example 4 *Figure 4 shows the derivation relation among 3 axioms. Particularly, α derives β, β dervies γ and γ also derives β. It is apparent that $R^*(\{\alpha\}) = \{\alpha, \beta, \gamma\}$. Their counting are also shown.*

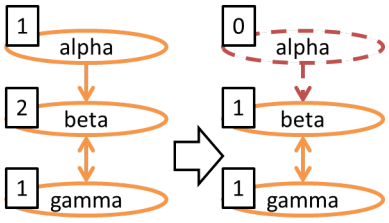

Figure 4: Incorrectness of Counting

If $Del = \{\alpha\}$, the only original axiom, then the updated closure should be empty. However, using the counting approach we will obtain $N(\alpha) = 0$, $N(\beta) = 1$ and $N(\gamma) = 1$, i.e. β and γ will be preserved.

The reason of such incorrectness is that when increasing the counting, the algorithm can not distinguish if the new derivation requires premise that is actually derived previously from the consequence or not.

(b) Similarly, this approach can not guarantee the completeness of its results. Below is an example:

Example 5 *Figure 5 shows the derivation relation among 4 axioms. Particularly, α derives both β and δ, which together derives γ. Apparently $R^*(\{\alpha, \gamma\}) = \{\alpha, \beta, \delta, \gamma\}$ and $N(\alpha) = N(\beta) = N(\delta) = 1$ and $N(\gamma) = 2$.*

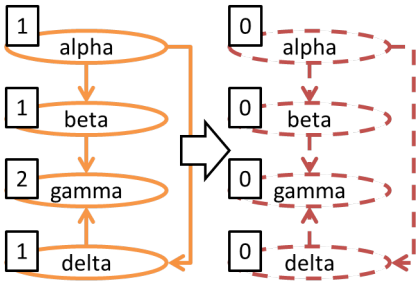

Figure 5: Incompleteness of Counting

Now if $Del = \{\alpha\}$, then the remaining γ should still be preserved. However using the counting approach we will obtain $N(\alpha) = N(\beta) = N(\delta) = N(\gamma) = 0$, i.e. γ will be incorrectly removed.

The reason of such incompleteness is that when decreasing the counting, the algorithm can not distinguish if the removed premises are used as premises in the same rule execution or

not. For example, in Figure 5, β and δ are used in the same rule execution to derive γ. However, $N(\gamma)$ will be decreased twice for the removal of β and δ, although γ only loses 1 derivation.

(c) This approach imposes even heavier overhead in reasoning than the global re-derivation approaches. On the one hand, in order to obtain the accurate counting, this approach has to exhaust all possible ways to derive each entailment. While in global re-derivation and local re-derivation approaches, the algorithm only needs to find one way to derive each entailment. On the other hand, if the reasoner wants to overcome the incompleteness and incorrectness problems mentioned above, it has to maintain all the possible rule executions for each entailment. This essentially computes all justifications of all results and is expensive even for simple DLs. For example, computing one justification for an entailment in $\mathcal{EL}$ is tractable, but computing all justifications for an entailment in $\mathcal{EL}$ is intractable. Hence, although reasoning is completely avoided when dealing with removal of entailments, reasoning in general is harder when deriving new entailments in this approach. Otherwise, the stream reasoning results are not guaranteed correct or complete.

In the K-Drive project, we want to develop a stream reasoning approach that addresses the above limitations of existing works. The motivations to our approach presented in the next section are:

1. We don't want to increase the complexity of stream reasoning in comparison to naive reasoning. Therefore we do not use an all-justification approach.

2. We want the results to be exactly the same as naive reasoning. Therefore we do not use the counting strategy. Combined with the above motivation, we will still use DRed instead of the the no re-derivation strategy.

3. We want our approach to be parallelisable so that it is applicable to the non-bookkeeping approach, when the deleted axioms are not in side conditions.

4. We want our approach to be applicable to bookkeeping approaches as well to deal with removal of any axiom.

5. Existing DRed approaches process unnecessary axioms in the list when doing re-derivation, either in a global scale, or within certain contexts, because their re-derivation starts from the remaining axioms, whose entailments may have been preserved. To address this issue, we will start from the over-deleted axioms, so that only the axioms that can contribute to the re-derivation will be involved.

4 Combining Forward And Backward Chaining

In order to avoid unnecessary axiom over-deletion and re-derivation, it is necessary to develop a re-derivation mechanism that focuses only on the preserved entailments that can be used to infer over-deleted axioms.

4.1 Full Backward Chaining Re-derivation

One way to achieve our goal is a full backward chaining procedure. Different from the forward chaining procedure that starts from the preserved axioms as in existing works, a backward chaining procedure

starts from the entailments that are attempted to be re-derived. It then checks which potential premises in the original closure can be used to derive such an entailment. If all the premises have been preserved during over-deletion or re-derived during re-derivation, then the target axiom can be re-derived. Otherwise, the algorithm can try to re-derive the potential premises. Iteratively, this procedure can re-derive all entailments that can be inferred from the preserved axioms.

Such a procedure can be described with the following algorithms:

Full Backward Chaining Re-derivation:
$fBCRD(L, S, R)$
INPUT: a list of axioms to be re-derived L, a partial closure S, a completion rule set R
OUTPUT: a set of re-derived axioms $Rederived$

1: $Rederived := \emptyset$
2: **while** $L \neq \emptyset$ **do**
3: get $\alpha \in L$
4: $L := L \setminus \{\alpha\}$
5: $fTest(L, S, Rederived, \{\alpha\}, \alpha, R)$
6: **return** $Rederived$

Given an closure after over-deletion S, a list of over-deleted axioms L and a rule set R, Algorithm $fBCRD(L, S, R)$ finds out all axioms in L that can be directly or indirectly re-derived from S:

1. In Step-1 it first initialises the set of re-derived entailments, which is empty initially.
2. From Step-2 to Step-5, it iteratively tests each entailment $\alpha \in L$ until it is empty. Such a testing is performed by a sub-procedure Algorithm *fTest*.

Full Test:
$fTest(L, S, Rederived, Testing, \beta, R)$
INPUT: a list of axioms to be re-derived L, a partial closure S, a set of re-derived axioms $Rederived$, a set of axioms being tested $Testing$, an axiom to be tested β and a set of completion rules R
OUTPUT: nothing, but L and $Rederived$ will be altered during the execution of the algorithm

1: **if** $\beta \notin Rederived$ **then**
2: **for** each $rule \in R$ **do**
3: **if** β can be used as the consequence of $rule$, and all premises $A = \{\alpha_1, \ldots, \alpha_n\}$ of $rule$ are in $S \cup L \cup Rederived$ **then**
4: **while** $A \cap L \neq \emptyset$ **do**
5: get $\alpha \in A \cap L$
6: $Test(L, S, Rederived, Testing \cup \{\beta\}, \alpha, R)$
7: **if** $A \subseteq S \cup Rederived$ **then**
8: $Rederived := Rederived \cup \{\alpha\}$
9: $L := L \setminus \{\beta\}$
10: **return**

Given S, L, $Rederived$, R and a set of axioms being tested $Testing$, Algorithm *fTest* will check if an entailment β can be re-derived with R from entailments in S. If β can be re-derived, the algorithm will extend $Rederived$ accordingly. To test the possibility of re-derivation, the algorithm will recursively test the premises of β in L:

1. The algorithm looks for possible rule execution that infers β. The consequence of such a rule execution should include β, and the premises A of such a rule execution should all be included in $S \cup L \cup Rederived$. $S \cup L \cup Rederived$. Note that S are entailments that have been preserved, $Rederived$ are the entailments that have been re-derived and L are the entailments that have not been tested yet, which means they can potentially be re-derived. If any premise α does not belong to $S \cup L \cup Rederived$, then there are the following possibilities:

   (a) α is not in the original completion closure, which indicates that the rule execution can not even be executed in the original closure. Hence, it is not possible to execute it to re-derive β with the remaining entailments.

   (b) α has been tested, but not included into $Rederived$, which indicates that α cannot be re-deirved from S. Hence, β cannot be re-derived using α.

   (c) α is in $Testing$, which means that the testing of $beta$ is recursively invoked by the testing of α. This indicates that β is used as a direct or indirect premise to re-derive α, hence one can no longer use α as a premise to re-derive β. Otherwise, the termination of the algorithm cannot be guaranteed.

2. Once such a rule execution is identified, the algorithm examines each premise $\alpha \in A$ that is still in the L, i.e. over-deleted but not tested yet. Step-6 tests the re-derivation of α by recursively invoking algorithm *fTest*. In this case, the set of axioms being tested is extended with α itself so that α won't be used as a premise when testing itself or its premises. Note that β has already been included in $Testing$. If α can be re-derived, it will be included into $Rederived$ during the recursive tests.

3. After all un-tested premises have been tested, Step-7 checks if all the premises have been either preserved, or re-derived.

4. If the above case is true, the algorithm will move β from L to $Rederived$. The change of L is redundant if $fTest(L, S, Rederived, Testing, \beta, R)$ is directly called by $fBCRD(L, S, R)$. But they are needed to ensure that axioms re-derived during recursively testing won't be tested again.

Let $R^*(\mathcal{O})$ be a completion closure of ontology $\mathcal{O}$ w.r.t. rule set R and OD be the set of over-deleted axioms, with the above algorithms, re-derivation can be performed with $fBCRD(OD, R^*(\mathcal{O}) \setminus OD, R)$. It can be shown that the above procedure produces the correct and complete re-derivation results:

Lemma 3 *Let $\mathcal{O}$ be an ontology, $S = R^*(\mathcal{O}) = FCC(\mathcal{O}, \emptyset, R)$ be the completion closure of $\mathcal{O}$ w.r.t a set of completion rules R, $Del \subseteq \mathcal{O}$ be a set of deleted axioms, $OD \subseteq S$ be a valid over-deletion w.r.t. Del, then $(S \setminus OD) \cup fBCRD(OD, S \setminus OD, R) = R^*(\mathcal{O} \setminus Del)$.*

Proof First of all, we show that $R^*(S \setminus OD) = R^*(\mathcal{O} \setminus Del)$:

1. The $\rightarrow$ direction: For any $\alpha \in S \setminus OD$, it has some justification $\mathcal{J}_O(\alpha) \subseteq \mathcal{O} \setminus Del$. Otherwise, according to the first condition in definition of valid over-deletion (Def. 2), $\alpha \in OD$. Hence, $S \setminus OD \subseteq R^*(\mathcal{O} \setminus Del)$. Following property (1) in Lemma 2, we have $R^*(S \setminus OD) \subseteq R^*(R^*(\mathcal{O} \setminus Del)) = R^*(\mathcal{O} \setminus Del)$.

2. The $\leftarrow$ direction: For any $\alpha \in R^*(O \setminus Del)$, according to the second condition in Def. 2, there is some justification of α that is not in OD, which means that α can be derived from $S \setminus OD$. Hence, $\alpha \in R^*(S \setminus OD)$.

With the above equation, we only need to show that $(S \setminus OD) \cup fBCRD(OD, S \setminus OD, R) = R^*(S \setminus OD)$:

1. The $\rightarrow$ direction is trivial since all axioms in $fBCRD(OD, S, R)$ are consequences of axioms in $S \setminus OD$: according to Step-7 of *fTest*, they are derived immediately from premise axioms that are either in $S \setminus OD$, or re-derived from axioms in $S \setminus OD$.

2. The $\leftarrow$ direction can also be shown inductively:

   (a) First of all, all axioms in $R(S \setminus OD)$ can be re-derived since their premises are preserved during the over-deletion;

   (b) Secondly, for any $\beta \in R^2(S \setminus OD) \setminus R(S \setminus OD)$, it will be tested by Step-5 of Algorithm *fBCRD*. If the testing is earlier than the testings of all its premises (which are all in $R(S \setminus OD)$), then β can be re-derived after its premises are recursively tested. The testing of β is after α is tested, then obviously β can use α as a premise to re-derive. Note that in either case, it is possible to use α as a premise to re-derive β because testing of β is directly invoked by Algorithm *fBCRD* hence α will not be included in $Testing$. Together we show that all axioms in $R^2(S \setminus OD) \subseteq (S \setminus OD) \cup fBCRD(OD, S, R)$.

   Inductively, we can show that $R^*(S \setminus OD) \subseteq (S \setminus OD) \cup fBCRD(OD, S, R)$.

Together, the lemma is proved. □

4.2 Combined Forward and Backward Chaining Re-derivation

The full backward chaining approach introduced in the previous subsection can be further optimised. Particularly, in the presented procedure, the testing of the same axiom may be invoked multiple times. Considering the following worst case example:

Example 6 *There are axioms $\alpha_1, \ldots, \alpha_n$ and a rule set R s.t. for each $i = 1, \ldots, n$, we have $R(\{\alpha_i\}) = \{\alpha_1, \ldots, \alpha_n\}$.*

Now assume $\alpha_1, \ldots, \alpha_n$ have all been over-deleted from a closure, and none of the preserved entailments can re-derive any of them. Algorithm fBCRD *will still try to re-derive them.*

For each α_i, since it can be inferred from any other α_j, Algorithm fTest *will attempt to recursively test each of the other α_j. During the testing of this α_j,* fTest *will further recursively test each of the third α_k. Such recursion continues until all $\alpha_1, \ldots, \alpha_n$ have been included in $Testing$.*

Together, it will test all the n entailments for $n!$ times.

Redundant tests imply repeated failed tests. In this section, we present a more efficient variant of the previous procedure that eliminates the redundant testings. The key-point is to combine forward and backward chaining in re-derivation:

1. Assuming we have a completion closure $S = R^*(\mathcal{O})$ and a set of over-deleted axioms OD, the purpose of re-derivation is to compute $R^*(S \setminus OD)$.

2. Forward chaining re-derivation achieves this by computing $FCC(S \setminus OD, \emptyset, R)$ either globally or locally, and may process unnecessary entailments.

3. Instead, we only need to find $R(S \setminus OD)$, and then compute $FCC(R(S \setminus OD), S \setminus OD, R)$. Since we have $R(S \setminus OD) \subseteq (S \setminus OD) \cup R(S \setminus OD)$, according to Lemma 2, results of $FCC(R(S \setminus OD), S \setminus OD, R)$ is the same as $R^*((S \setminus OD) \cup R(S \setminus OD))$. According to Lemma 1, $R^*((S \setminus OD) \cup R(S \setminus OD)) = R^*(S \setminus OD)$.

The above procedure is the forward chaining part. The $R(S \setminus OD)$ will be computed by backward chaining. It can be achieved with a procedure similar to $fBCRD$:

Backward Chaining Re-derivation:
$BCRD(L, S, R)$
INPUT: a list of axioms to be re-derived L, a partial closure S, a completion rule set R
OUTPUT: a set of re-derived axioms $Rederived$

1: $Rederived := \emptyset$
2: **for** each $\alpha \in L$ **do**
3: $\quad Test(S, Rederived, \alpha, R)$
4: **return** $Rederived$

Test:
$Test(S, Rederived, \beta, R)$
INPUT: a partial closure S, a set of re-derived axioms $Rederived$, an axiom to be tested β and a set of completion rules R
OUTPUT: nothing, but $Rederived$ will be altered during the execution of the algorithm

1: **for** each $rule \in R$ **do**
2: $\quad$ **if** β can be used as the consequence of $rule$, and all premises $A = \{\alpha_1, \ldots, \alpha_n\}$ of $rule$ are in S **then**
3: $\quad\quad Rederived := Rederived \cup \{\beta\}$
4: $\quad\quad$ **return**

As we can see, the procedure is different from the previous full backward chaining re-derivation on the following aspects:

1. Instead of testing axioms with algorithm *fTest*, a new algorithm *Test* is used.
2. *Test* no longer recursively checks if a premise is re-derivable when it is not immediately available in S. Instead, it only checks if all premises are in S, which is the preserved closure. Hence the set $Testing$ is not needed, because a tested axiom will not be used as premise to re-derive another tested axiom.
3. As a consequence $BCRD(L, S, R)$ will compute all axioms in L that can be directly re-derived from S.

Therefore, for a completion closure S and a valid over-deletion OD, we have $R(S \setminus OD) = BCRD(OD, S \setminus OD, R)$. Combining with the forward chaining part mentioned above, re-derivation of $R^*(S \setminus OD)$ can be achieved:

Lemma 4 *Let $\mathcal{O}$ be an ontology, $S = R^*(\mathcal{O}) = FCC(\mathcal{O}, \emptyset, R)$ be the completion closure of $\mathcal{O}$ w.r.t. a set of completion rules R, $Del \subseteq \mathcal{O}$ be a set of deleted axioms, $OD \subseteq S$ be a valid over-deletion w.r.t. Del, then $R^*(\mathcal{O} \setminus Del) = FCC(BCRD(OD, S \setminus OD, R), S \setminus OD, R)$.*

Proof As shown in the proof of Lemma 3, we have $R^*(\mathcal{O} \setminus Del) = R^*(S \setminus OD)$. Hence we only need to prove $R^*(S \setminus OD) = FCC(BCRD(OD, S \setminus OD, R), S \setminus OD, R)$. According to Lemma 2, we only need to prove that $R(S \setminus OD) \subseteq (S \setminus OD) \cup BCRD(OD, S \setminus OD, R)$.

For each $\alpha \in R(S \setminus OD)$, we omit the trivial situation that $\alpha \in S \setminus OD$. Since S is a completion closure, it is obvious that $\alpha \in OD$. Therefore, α will be tested in $Test(S \setminus OD, Rederived, \alpha, R)$ in Step-3 of $BCRD(OD, S \setminus OD, R)$. Due to the fact that $\alpha \in R(S \setminus OD)$, α can be re-derived from premises in $S \setminus OD$ by some rule in R. According to Step-2 of Algorithm *Test*, α will be added into $Rederived$. Hence $\alpha \in BCRD(OD, S \setminus OD, R)$ This shows that $R(S \setminus OD) \subseteq (S \setminus OD) \cup BCRD(OD, S \setminus OD, R)$ and proves the lemma. □

When completion rule set R is tractable, this procedure is also tractable since both $BCRD$ and FCC will be tractable. Particularly, in $BCRD$, every axiom in OD will be tested at most once. And in each testing, the algorithm will look for backward chaining premises only from axioms in $R^*(Q) \setminus OD$. This suggests that the backward-chaining re-derivation itself does not affect the tractability of reasoning in general.

Conceptually, such a combined forward and backward chaining re-derivation is different from the forward chaining re-derivation and full backward chaining re-derivation on the following aspects:

1. **Minimal problem space:** the backward-chaining algorithm only attempts to re-derive the entailments that have been over-deleted. The remaining entailments will not even be tested;

2. **Small search space:** an entailment can be derived by many different rule executions. The algorithms presented above will only look into those rule execution whose premises are all preserved. And once an entailment is re-derived, other possible rule executions will not be examined.

3. **Target-oriented:** among the preserved entailments, only those who can potentially contribute to the re-derivation will be examined in the backward chaining stage. Preserved entailments that do not contribute will not be involved at all. Similarly, in the forward chaining stage of re-derivation, this approach will only process re-derived axioms. Preserved axioms will not be processed.

These differences make the combined forward and backward chaining re-derivation more efficient than the forward chaining or full backward chaining re-derivation, especially when the over-deleted entailments are much less than the preserved entailments. In the evaluation, we will empirically compare our approach to existing ones and show that the proposed re-derivation approach can indeed improve over-deletion, re-derivation and stream reasoning efficiency in general.

Our approach also does not rely on bookkeeping dependencies between premise and consequence axioms. When performing Step-2 of Algorithm *Test*, an implemented system only needs to identify one candidate premise α, and then it can use α in the same way as in Step-5 of Algorithm *FCC* to find other premises. The identification of α can be realised by exploiting the structural relationships between premise and consequence of each rule. For example, in order to re-derive $C \sqsubseteq E$ with backward chaining of rule $\mathbf{R}_{\sqsubseteq}$, the reasoner only needs to search for a preserved entailment $C \sqsubseteq D$ s.t. $D \sqsubseteq E$ is also preserved. To re-derive $E \sqsubseteq \exists S.D$ with backward chaining of rule $\mathbf{R}_{\exists}$, the reasoner only needs to search of a preserved entailment $E \sqsubseteq \exists R.C$ s.t. $D \sqsubseteq D$ is preserved and $R \sqsubseteq^* S$ holds. In general, if *FCC* can be performed without bookkeeping, our approach can be performed without bookkeeping.

4.3 Improving Existing Approaches

Since our approach focuses on efficient re-derivation, it can be used to substitute the re-derivation components of existing approaches and improve their performance:

1. *Bookkeeping DRed Approaches:* In our approach, bookkeeping can be easily implemented when performing the testing of over-deleted axioms in algorithm *Test*. Particularly, when an axiom is re-derived with backward chaining, same bookkeeping can be generated as if it was derived with forward chaining from the same premise axioms and side conditions. Eventually, the bookkeeping information will be the same as in existing works and the performance and memory overhead should also be comparable. In our evaluation, we actually implemented the TMS-based approach [Ren and Pan, 2011] with our re-derivation technique and tested its performance. By replacing the global re-derivation $FCC(R^*(\mathcal{O}) \setminus OD, \emptyset, R)$ with the combined re-derivation, less axioms need to be processed and performance can be improved.

2. *Non-bookkeeping DRed Approach:* Our approach is applicable to consequence-based algorithms that can be implemented in parallel since it is also parallelisable. Particularly, in algorithm *BCRD*, the testings of individual over-deleted axioms are completely independent from one another and rely only on the original input. Hence the testings can be implemented in parallel. The forward chaining stage of re-derivation is using the same algorithm for computing the closure hence can also be parallelised in the same way.

   More importantly, our approach also helps to address the unnecessary axiom re-derivation problem of the existing non-bookkeeping DRed Approach. The DEL set is already a valid over-deletion and hence can be used to replace OD_L in our approach. Since $BCRD(DEL, R^*(\mathcal{O}), R) \subseteq DEL \subseteq OD_L$, it is clear that $FCC(BCRD(DEL, R^*(\mathcal{O}), R), R^*(\mathcal{O}) \setminus DEL, R)$ can avoid re-deriving and re-processing the unnecessarily deleted axioms in OD_L.

3. *Counting Approach:* As discussed earlier, applying counting approach can be incomplete if a single derivation involves multiple premise axioms. This problem can be solved by performing re-derivation with our approach. Firstly, counting can be used to identify the set of entailments to be deleted. Then such a set of axioms can be used as the over-deletion to perform re-derivation with our approach. The algorithm *Test* should be modified accordingly such that it does not terminate when the axiom β is re-derived. Instead, it continues the re-derivation and increases the counting when alternative derivation is found.

   It is worth mentioning that our approach can be implemented with ease in DynamiTE [Urbani et al., 2013], the parallel materialisation and query answering engine that implements the counting approach. Particularly, given a consequence β, the searching for a premise A in a the preserved closure S w.r.t. a completion rule can be easily implemented as a query that can be efficiently executed by DynamiTE.

Due to the capability to be integrated with and to improve existing approaches, the new stream reasoning technique presented in this deliverable can be used together with existing successful technologies to realise the stream maintenance and reasoning framework presented in D6.1, and to support different forms of ontology stream dynamics and different types of stream reasoning services.

5 Experimental Evaluation

In order to evaluate the usefulness and performance of our approach, we conducted empirical evaluation to find out:

1. Whether our approach can be used to reduce the number of axioms to be over-deleted and/or processed in re-derivation, in comparison to global re-derivation and local re-derivation.

2. Whether our approach can be used to achieve efficient stream reasoning, in comparison to naive re-computation of all results.

5.1 Implementation and Benchmark

We implemented the combined forward and backward chaining approach to re-derivation as presented in Sec. 4:

1. In order to compare with the non-bookkeeping DRed approach [Kazakov and Klinov, 2013], we first implemented a consequence-based algorithm with context. This algorithm is used by the parallel $\mathcal{EL}^{++}$ reasoner ELK [Kazakov et al., 2013]. On this implementation, we also implemented the forward-chaining over-deletion used in the non-bookkeeping DRed approach. As introduced in Sec. 3, such an approach first obtains a set DEL that consists of all entailments that can be derived from the deleted original axioms. It then effectively over-deletes and re-derives all non-original GCIs with the same context as some axiom in DEL.

2. In order to compare with the bookkeeping DRed approach, a separate but identical implemented was augmented with the TMS mechanism proposed by our previous work [Ren and Pan, 2011]. With this mechanism, we also implemented the TMS-based over-deletion used by their approach.

3. In order to support reasoning with our evaluation benchmark, our implementations were extended with ABox completion rules. Implementation-wise, this was achieved by internalising ABox axioms with TBox axioms. Such a treatment does not affect the completeness of results on our evaluation benchmark.

4. In order to support the DL used by our evaluation benchmark, our implementations were also extended with the following additional rule to exploit inverse roles in ABox reasoning:

$$\mathbf{R}_{\mathcal{I}} \frac{(a,b) : r, a : C}{b : \exists s.C} : \exists s.C \text{ occurs in } \mathcal{O}, r \sqsubseteq^{*,-} s$$

   where $r \sqsubseteq^{*,-} s$ if $r \equiv s^- \in \mathcal{O}$ or $r' \sqsubseteq^* r'$, $r \sqsubseteq^{*,-} s'$ and $s' \sqsubseteq^* s$ and $r \sqsubseteq^* s$ if $r \sqsubseteq s \in \mathcal{O}$, or $r \sqsubseteq^* t$ and $t \sqsubseteq^* s$, or $r \sqsubseteq^{*,-} t$ and $t \sqsubseteq^{*,-} s$. With such extension, the rule set is tractable but it is complete for our evaluation benchmark. Note that in the above formulation, all premise axioms still share a context $\{a\}$. Hence, the extention does not affect the parallelisability of the original rule set.

For evaluation, we executed the implementations to compute the atomic concept hierarchies and the atomic types of each individual of an ontology and update results with stream reasoning. As we introduced in Sec. 2, all these results can be achieved by computing the a completion closure.

For these experiments, we prepared the evaluation benchmark using the Lehigh University Benchmark (LUBM) [Guo, 2005]. As we mentioned earlier, our implementation is complete for reasoning of this ontology. The motivation of using the LUBM is that the LUBM is a synthetic benchmark that can generate arbitriraly large set of data. Such data usually contains common patterns of axioms that re-occur frequently in different snapshots of the stream. This is similar to the public data set we are dealing with in our case studies. Our evaluation benchmark was generated as follows:

1. We used LUBM to generate data files of 10 universities.

2. We randomly separated all axioms in these 10 universities into 200 ABoxes, denoted by $A_1, \ldots, A_{200}$.

3. For each test, we started reasoning with the LUBM TBox and ABoxes $A_1, \ldots, A_{50}$, i.e. about 2.5 universities.

4. When updating, we removed $1 \leq n \leq 50$ ABoxes and added another n ABoxes. Such updating was repeated for $\lfloor \frac{150}{n} \rfloor$ times.

All experiments were conducted in an environment of 64-bit Ubuntu 13.10 with 3.20GHz CPU and 10G RAM allocated to JVM.

5.2 Evaluating the Re-derivation

To examine if our approach can reduce the number of over-deleted and/or processed axioms in re-derivation, for each test, we were interested in the sizes of the following sets:

1. Del: deleted original axioms.

2. R^*: the completion closure.

3. OD_{TMS}: the over-deleted axioms in the TMS-based approach.

   These are also the axioms to be processed in the backward chaining stage if our re-derivation approach is applied with the TMS-based approach.

4. $BCRD_{TMS}$: the axioms re-derived in the backward chaining stage of our re-derivation approach when applied with the TMS-base approach.

   These are also the axioms to be initialised in L in the forward chaining stage of our approach.

   As a comparison, $R^* \setminus OD_{TMS}$ are the axioms to be initialised in the list in global re-derivation.

5. DEL: the axioms directly or indirectly inferred from Del axioms with the forward chaining over-deletion in the non-bookkeeping approach.

   These are also the axioms to be processed in the backward chaining stage if our re-derivation approach is applied with the non-bookkeeping approach.

6. OD_L: the non-original axioms withint the same context as some axioms in DEL. These axioms, even if preserved, will be re-derived by the forward chaining re-derivation of the non-bookkeeping approach. Hence, they can be regarded as the effectively over-deleted axioms with this approach.

7. $BCRD_L$: the axioms re-derived in the backward chaining stage of our re-derivation approach when applied with the non-bookkeeping approach and using DEL as the over-deletion.

   These are also the axioms to be initialised in list L in the forward chaining stage of our approach.

We conducted the experiments for $n = 1, 2, 5, 10$, i.e. 2%, 4%, 10% and 20% of the ABox were updated respectively. For each n, the size of above sets were obtained on the $\lfloor \frac{150}{n} \rfloor$ runs. The average percentages of $|Del|$, $|R^* \setminus OD_{TMS}|$, $|BCRD_{TMS}|$, $|DEL|$, $|OD_L|$, $|BCRD_L|$ against $|R^*|$ are illustrated in Tabel 3 and Figure 6.

There are several observations we can make from the results:

1. With the growing of Del, $BCRD_{TMS}$ is growing and $R^* \setminus OD_{TMS}$ is decreasing. When Del is low, as shown in Table 3 and Fig. 6, $BCRD_{TMS}$ is much much smaller than $R^* \setminus OD_{TMS}$ (e.g. 0.17% v.s. 98.80% when Del is 0.42% of R^*), indicating that the forward chaining stage in

Table 3: Re-derivation Evaluation Results.

$\frac{n}{50}$	2%	4%	10%	20%
$\lvert Del \rvert$	0.42%	0.85%	2.12%	4.23%
$\lvert OD_{TMS} \rvert$	1.20%	2.40%	5.95%	11.74%
$\lvert R^* \setminus OD_{TMS} \rvert$	98.80%	97.60%	94.05%	88.26%
$\lvert BCRD_{TMS} \rvert$	0.17%	0.33%	0.77%	1.37%
$\lvert DEL \rvert$	3.39%	5.89%	12.62%	22.22%
$\lvert OD_L \rvert$	5.52%	9.45%	19.58%	32.83%
$\lvert BCRD_L \rvert$	2.24%	3.80%	7.93%	13.68%

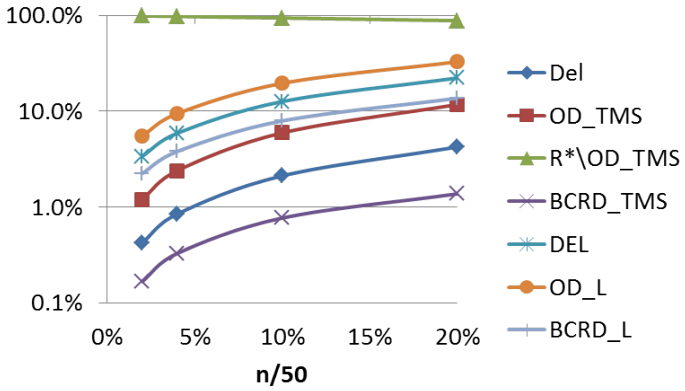

Figure 6: Re-derivation Evaluation Results

our combined approach processes much much less axioms than the global re-derivation. Even when taking into account the cost of the backward chaining, as implied by the size of OD_{TMS}, our combined forward and backward chaining approach should still process less axioms than the TMS-based approach. This indicates that our approach is more efficient than the global re-derivation DRed when only a small fragment of the ontology is removed during update.

Nevertheless, it should be expected that when n is big enough, $R^* \setminus OD_{TMS}$ will be smaller than $BCRD_{TMS}$. In this case, using a global re-derivation might be more beneficial than our approach.

2. Regardless of the percentage of n, OD_{TMS} is always smaller than DEL (e.g. 1.20% v.s. 3.39% when Del is 0.42% of R^*), which is about 60% of the size of OD_L. This indicates that the context-based approach usually unnecessarily over-deletes many more unnecessary entailments than the TMS-based approach. For example, when 20% of the ontology axioms are removed, a TMS-based approach over-deletes in average 11.74% of the entailments in R^*. While the context-based non-bookkeeping approach over-deletes in average 32.83% of the entailments. Note that the gap between the two figures will be re-derived during the re-derivation stage. By applying our re-derivation approach, the over-deletion in non-bookkeeping approach can be reduced, i.e. over-deleting DEL instead of OD_L. As a consequence, unnecessary re-derivation can be avoided.

It is also worth noting that $BCRD_{TMS}$ is much smaller than $BCRD_L$, about 10% in size. This shows that our approach synergises better with the TMS-based over-deletion than with the non-bookkeeping over-deletion, performing much less backward chaining and forward chaining re-derivations. This is mainly due to the fact that OD_{TMS} is smaller than DEL.

3. When the deletion is relatively small, even DEL and $BCRD_L$ are also very small. Taking the TMS-based over-deletion as an example, when 2% of the ABox are deleted, only 1.2% of all the entailments are over-deleted, among which only 0.17% of the entailments are needed to initiate the forward chaining stage of re-derivation. This indicates that the re-derivation effort is much smaller than naive re-computation. As we will show in the next evaluation. This also implies that our approach should not impose significant performance overhead when applied with the counting-based over-deletion approach.

To summarise, our combined forward and backward chaining re-derivation technology is very suitable for ontology updating with small scale deletion. It can significantly reduce the re-derivation effort in comparison to the bookkeeping global re-derivation approach. It can reduce the unnecessary over-deletion in comparison to the non-bookkeeping local re-derivation approach. It can also be used to address the completeness issue of the counting approach.

5.3 Evaluating Stream Reasoning

To examine if our approach can be used to support efficient stream reasoning, we also compared the performance of stream reasoning and naive re-computation. From the previous evaluation, we noticed that TMS-based approach over-deletes less axioms than context-based approach. Hence in this evaluation we used the implementation with TMS to perform stream reasoning. To have a better understanding about the usefulness of the solution, we also explored the performance and memory overhead of the TMS. Naive re-computation was performed by the implementation without TMS. In this evaluation, we were particularly interested in the following time figures:

1. $T_{initial}$: the naive reasoning time of the TMS reasoner. This can be regarded as the start-up time of the stream reasoner when it encounters a new ontology.

2. M_{TMS}: the naive reasoning memory consumption of the TMS reasoner. This can be regarded as the memory required to perform reasoning with a TMS.

3. T_{naive}: the naive reasoning time of the non-TMS reasoner.

4. M_{naive}: the navie reasoning memory consumption of the non-TMS reasoner.

5. T_{stream}: the stream reasoning time of the TMS reasoner when the ontology is updated. It consists of the following components:

   (a) $T_{deletion}$: the deletion time, including over-deletion and re-derivation.

   (b) $T_{addition}$: the addition time, i.e. time of incremental reasoning.

With the above figures, we define the following percentages:

$$\%_{initial} = \frac{T_{initial}}{T_{naive}} \times 100\%$$
$$\%_{memory} = \frac{M_{TMS}}{M_{naive}} \times 100\%$$
$$\%_{deletion} = \frac{T_{deletion}}{T_{naive}} \times 100\%$$
$$\%_{addition} = \frac{T_{addition}}{T_{naive}} \times 100\%$$

$\%_{initial}$ implies the overhead of TMS reasoner in comparison to a non-TMS reasoner. $\%_{memory}$ implies the memory overhead of a TMS reasoner in comparison to a non-TMS reasoner. Note that the two reasoners are using the same rule set. The only difference is the on-the-fly construction of TMS. When $\%_{deletion} + \%_{addition} \leq 100\%$, stream reasoning will be more efficient than naive re-computation.

For $T_{initial}$, M_{TMS}, T_{naive} and M_{naive}, we performed tests for 151 times, on the ABoxes $A_1 \cup \cdots \cup A_{50}, A_2 \cup \cdots \cup A_{51}, \ldots, A_{151} \cup \cdots \cup A_{200}$. Each time, we calculated $\%_{initial}$ and $\%_{memory}$. For $T_{deletion}$ and $T_{addition}$, we conducted the experiments for $n = 1, 2, 5, 10$. For each n, the stream reasoning were performed for $\lfloor \frac{150}{n} \rfloor$ times. The reasoning output of the stream reasoner was the same as the naive reasoner. The average values of the above figures are illustrated in Table 4 and Figure 7.

Table 4: Stream Reasoning Evaluation Results

$\%_{initial}$	125.89%			
$\%_{memory}$	121.56%			
$\frac{n}{50}$	2%	4%	10%	20%
$\%_{deletion}$	7.37%	14.06%	37.12%	70.21%
$\%_{addition}$	5.94%	15.17%	33.87%	52.44%
$\%_{stream}$	13.31%	29.23%	70.99%	122.65%

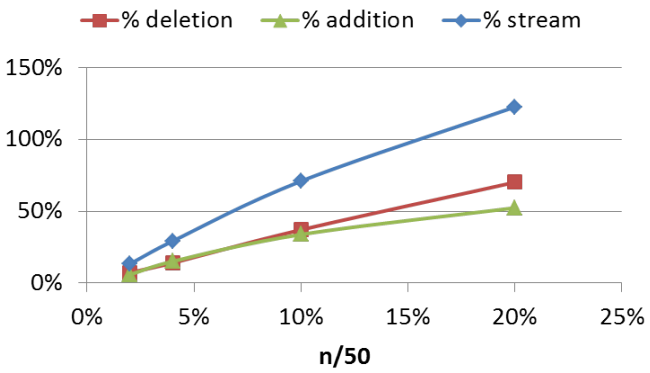

Figure 7: Stream Reasoning Evaluation Results

From the above figures, we have the following observations:

1. Using TMS-based over-deletion will impose a performance and memory over-head. As the figures of $\%_{initial}$ and $\%_{memory}$ shows in Table 4, the reasoning time was about 25.89% longer than the same reasoner without TMS. And the TMS reasoner consumed 21.56% more memory. However, when dealing with ontology updates, stream reasoning with TMS-based over-deletion and our re-derivation approach can be more efficient than naive re-computation. Hence using TMS pays off when an ontology can be initially reasoned with off-line and is expected to be updated on-line. In that case, the benefits of stream reasoning out-weight the cost of constructing and maintaining the TMS.

2. Apparently, when n increases, the deletion and addition become larger and take more time to accomplish. Hence the smaller the update is, the more significant stream reasoning out-performs naive re-computation. For example, as the figure of $\%_{stream}$ shows in Table 4, when 2% of the ABox is updated, stream reasoning took in average 13.31% of the time of naive reasoning. This shows the usefulness of our approach. In many real-world applications such as ontology authoring, the updates are even much smaller than 2%, usually with only a few axioms. In such applications, stream reasoning should be much more efficient than naive re-computation.

3. When the update is large enough, naive re-computation out-performs stream reasoning. Judging from Figure 7, approximately 15% update is the turning point.

To summarise, our implementation and evaluation showed that our approach, together with TMS-based over-deletion, can perform efficient stream reasoning, and is particularly useful for scenarios where the ontology is frequently updated with small changes.

6 Conclusion

In this deliverable, we present an approach to ontology stream reasoning with combined forward and backward chaining completion. Particularly, in a DRed framework, our approach first uses backward chaining to re-derive part of the over-deleted axioms that can be directly inferred by preserved axioms, and then uses these directly re-derived axioms to initiate forward chaining and re-derive the completion closure of the preserved axioms. On the one hand, this approach exploits the fact that the backward chaining is restricted to using only preserved axioms to infer only over-deleted axioms, ensuring small problem space and search space. On the other hand, the combination of forward and backward chaining avoids the redundant backward chaining tests in a full backward chaining approach.

This approach is focused on the re-derivation stage of DRed. Hence it can be combined with different over-deletion techniques. Compared to existing works, this approach helps to reduce the unnecessary over-deletion and re-derivation. It can also be used to address the completeness issue in the counting approach. The implementation of our approach does not affect the parallelisation or tractability of reasoning and its mechanism is applicable to any consequence-based algorithm.

We conducted extensive evaluation on a benchmark generated with the LUBM. Results showed that our approach can indeed reduce unnecessary over-deletion and/or re-derivation in a DRed stream reasoner. An implementation with the TMS-based over-deletion also out-performed a naive re-computation implementation with update size up to approximately 15%. Our approach works particularly well when the ontology update is of small size in comparison to the ontology, which is where stream reasoning is mostly needed.

The backward chaining stage of our approach derives the immediate results of the preserved closure. Such an idea has also been exploited by Kazakov and Klinov [Kazakov and Klinov, 2013], particularly

in their Algorithm 4. The difference is that, they derive such immediate results by foward chaining with the un-deleted original axioms, which will essentially re-derive the entire broken contexts. Our approach uses backward chaining to avoid the unnecessary re-derivation of the preserved axioms in these contexts.

Backward chaining can be implemented easily with rule systems. This indicates that the original DRed strategy [Gupta et al., 1993] and the maintenance rule-based solution [Volz et al., 2005] can also exploit such a backward chaining idea. Nevertheless, we notice that backward chaining only needs to be performed to re-derive immeidate consequence. Hence, expensive recursive backward chaining can be avoided. Also, our approach only considers a given completion rule set and does not need to generate additional rules from the axioms.

In the future we would like to combine the strengths of different approaches to implement the stream maintenance and reasoning framework presented in D6.1. Particularly, we plan to use TMS to deal with deletion of side condition axioms and contexts to deal with non-side condition axioms. We also plan to switch among different re-derivation techniques based on the size of the updates. To achieve such a goal we will do more implementation and experiments to better understand the dynamics of pros and cons of different approaches.

Acknowledgment

This research has been funded by the European Commission within the 7th Framework Programme/Maria Curie Industry-Academia Partnerships and Pathways schema/PEOPLE Work Programme 2011 project K-Drive number 286348 (cf. `http://www.kdrive-project.eu`).

References

[Guo, 2005] (2005). LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):158 – 182.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

[Baader et al., 2005] Baader, F., Lutz, C., and Suntisrivaraporn, B. (2005). 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)*.

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

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

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

[Cuenca Grau et al., 2010] Cuenca Grau, B., Halaschek-Wiener, C., Kazakov, Y., and Suntisrivaraporn, B. (2010). Incremental classification of description logics ontologies. *J. Autom. Reason.*, 44:337–369.

[Gao et al., 2014] Gao, S., Scharrenbach, T., and Bernstein, A. (2014). The CLOCK Data-Aware Eviction Approach: Towards Processing Linked Data Streams with Limited Resources. In *Proceedings of the 11th Extended Semantic Web Conference*, pages 6–20. Springer.

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

[Horrocks et al., 2004] Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). SWRL: A Semantic Web Rule Language — Combining OWL and RuleML. W3C Member Submission, `http://www.w3.org/Submission/SWRL/`.

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

[Kazakov, 2009] Kazakov, Y. (2009). Consequence-Driven Reasoning for Horn SHIQ Ontologies. In *IJCAI 2009*.

[Kazakov and Klinov, 2013] Kazakov, Y. and Klinov, P. (2013). Incremental reasoning in owl el without bookkeeping. In *The Semantic Web–ISWC 2013*, pages 232–247. Springer.

[Kazakov et al., 2013] Kazakov, Y., Krötzsch, M., and Simančík, F. (2013). The incredible elk. *Journal of Automated Reasoning*, pages 1–61.

[Klarman and Meyer, 2013] Klarman, S. and Meyer, T. (2013). Prediction and explanation over dl-lite data streams. In *Proceedings of the International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-19)*.

[Krötzsch, 2012] Krötzsch, M. (2012). The not-so-easy task of computing class subsumptions in owl rl. In *The Semantic Web–ISWC 2012*, pages 279–294. Springer.

[Lecue and Pan., 2013] Lecue, F. and Pan., J. Z. (2013). Predicting Knowledge in An Ontology Stream. In *Proc. of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013).*

[Motik et al., 2008] Motik, B., Grau, B. C., Horrocks, I., Wu, Z., Fokoue, A., and Lutz, C. (2008). OWL 2 Web Ontology Language: Profiles. W3c working draft, W3C.

[Qi and Yang, 2008] Qi, G. and Yang, F. (2008). A survey of revision approaches in description logics. In *Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems*, RR '08, pages 74–88, Berlin, Heidelberg. Springer-Verlag.

[Ren and Pan, 2011] Ren, Y. and Pan, J. Z. (2011). Optimising ontology stream reasoning with truth maintenance system. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 831–836. ACM.

[Simančík et al., 2011] Simančík, F., Kazakov, Y., and Horrocks, I. (2011). Consequence-based reasoning beyond horn ontologies. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*, pages 1093–1098. AAAI Press.

[Suntisrivaraporn, 2008] Suntisrivaraporn, B. (2008). Module Extraction and Incremental Classification: A Pragmatic Approach for $\mathcal{EL}^+$ Ontologies. In *ESWC'08*.

[Urbani et al., 2013] Urbani, J., Margara, A., Jacobs, C., van Harmelen, F., and Bal, H. (2013). Dynamite: Parallel materialization of dynamic rdf data. In *The Semantic Web–ISWC 2013*, pages 657–672. Springer.

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