

Stream Reasoning For Linked Data

E. Della Valle & J.Z. Pan

Heraklion
Greece
May 29th – June 2nd
2011



[3] Stream Reasoning techniques for RDFS++

Emanuele Della Valle

emanuele.dellavalle@polimi.it

<http://emanueledellavalle.org>

- This work is licensed under the Creative Commons Attribution 3.0 Unported License.
- **Your are free:**
 -  **to Share** — to copy, distribute and transmit the work
 -  **to Remix** — to adapt the work
- **Under the following conditions**
 -  **Attribution** — You must attribute the work by inserting
 - “© streamreasoning.org” at the end of each reused slide
 - a credits slide stating
 - These slides are partially based on
“Streaming Reasoning for Linked Data 2011”
by Emanuele Della Valle and Jeff Z. Pan
<http://streamreasoning.org/sr4ld2011>
- To view a copy of this license, visit
<http://creativecommons.org/licenses/by/3.0/>

Agenda

Heraklion
Greece
May 29th – June 2nd
2011



- Introduction
- State-of-the-Art – the DRed Algorithm
- The Streaming Approach
- Conclusions

Stream Reasoning For Linked Data

E. Della Valle & J.Z. Pan

Heraklion
Greece
May 29th – June 2nd
2011



[3.1] Introduction

Emanuele Della Valle

emanuele.dellavalle@polimi.it

<http://emanueledellavalle.org>

- Introduction
 - Let's recall the Stream Reasoning slides
 - The problem
 - An Example in Social Media
- State-of-the-Art – the DRed Algorithm
- The Streaming Approach
- Conclusions

- Sensor networks, ...



- traffic engineering, ...



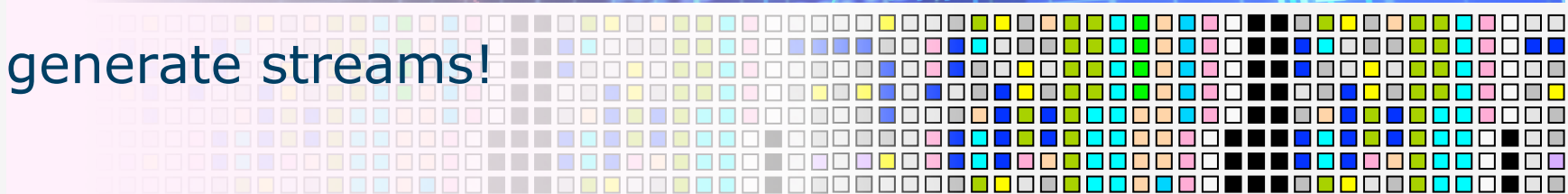
- social networking, ...



- financial markets, ...



- generate streams!



- Making sense
 - in **real time**
 - of *gigantic* and inevitably *noisy* **data streams**
 - in order to **support the decision process** of extremely *large numbers of concurrent user*

- The Semantic Web, the Web of Data **is doing fine**
 - RDF, RDF Schema, SPARQL, OWL, RIF
 - well understood theory,
 - rapid increase in scalability
- **BUT it pretends that the world is static**
or at best a low change rate
both in change-volume and change-frequency
 - ontology versioning
 - belief revision
 - time stamps on named graphs
- It sticks to the traditional **one-time semantics**

- Idea origination
 - Can **continuous semantics** be ported to **reasoning**?
 - This is an **unexplored yet high impact** research area!
- Stream Reasoning
 - ***Logical reasoning in real time on gigantic and inevitably noisy data streams in order to support the decision process of extremely large numbers of concurrent users.***
 - S. Ceri, E. Della Valle, F. van Harmelen and H. Stuckenschmidt, 2010
- Note: making sense of streams necessarily requires processing them against *rich background knowledge*

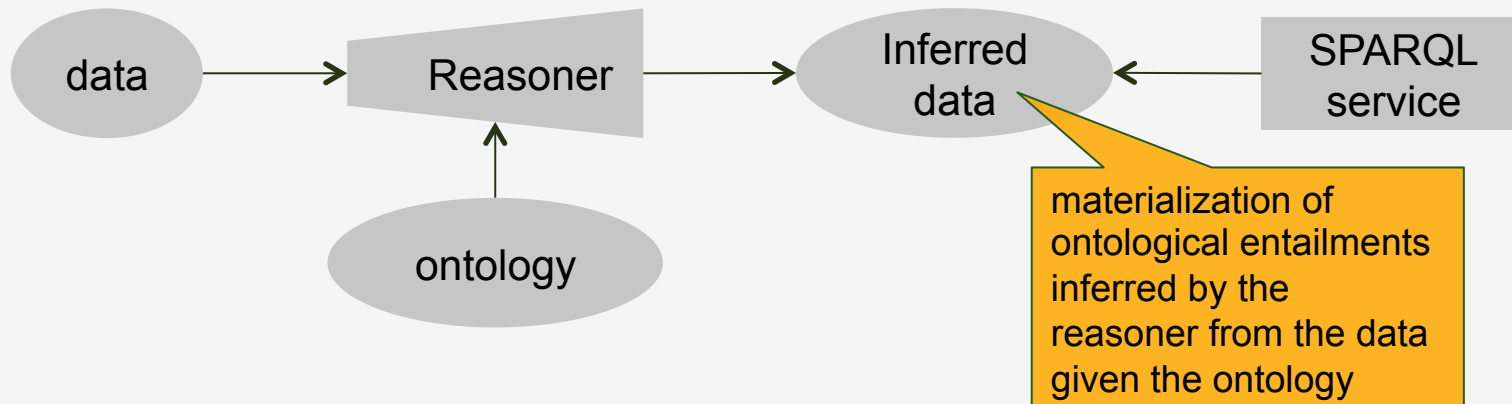
- Relation with data-stream systems
 - Just as RDF relates to data-base systems?
- Query languages for semantic streams
 - Just as SPARQL for RDF but with continuous semantics?
- Reasoning on Streams
 - Formal representations for stream reasoning
 - Notions of soundness and completeness
 - Efficiency
 - Scalability
- Dealing with incomplete & noisy data
 - Even more so than on the current Web of Data
- Distributed and parallel processing
 - Streams are parallel in nature

- Since 2007 we have been investigating
 - **RDF streams**
 - the natural extension of the RDF data model to the new continuous scenario and
 - **Continuous SPARQL (or simply C-SPARQL)**
 - the extension of SPARQL for querying RDF streams.
 - **Efficient incremental updates of deductive closures**
 - specifically considering the nature of data streams
 - **Effective inductive stream reasoning** (joint work with Siemens - Munich)
 - See paper in IEEE IS special issue on Social Media Analytics
 - **Architectures of a Stream Reasoner**

- SPARQL alone cannot answer queries that require reasoning

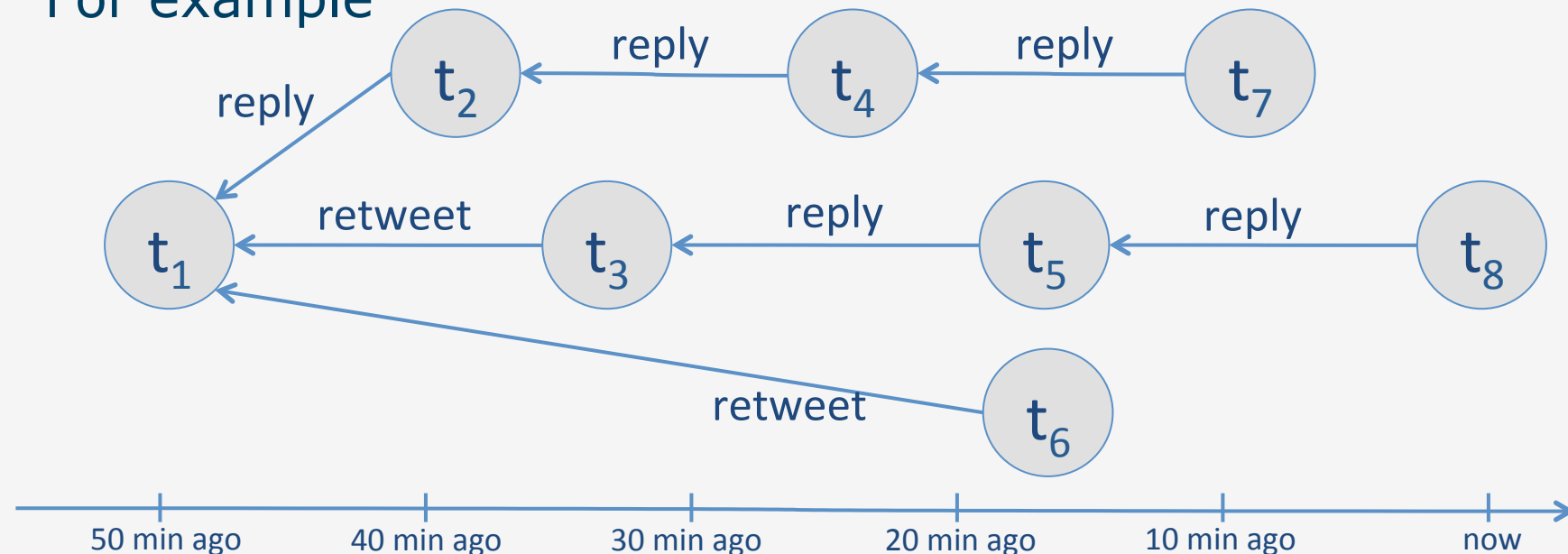


- but a reasoner can be exposed as a SPARQL service.

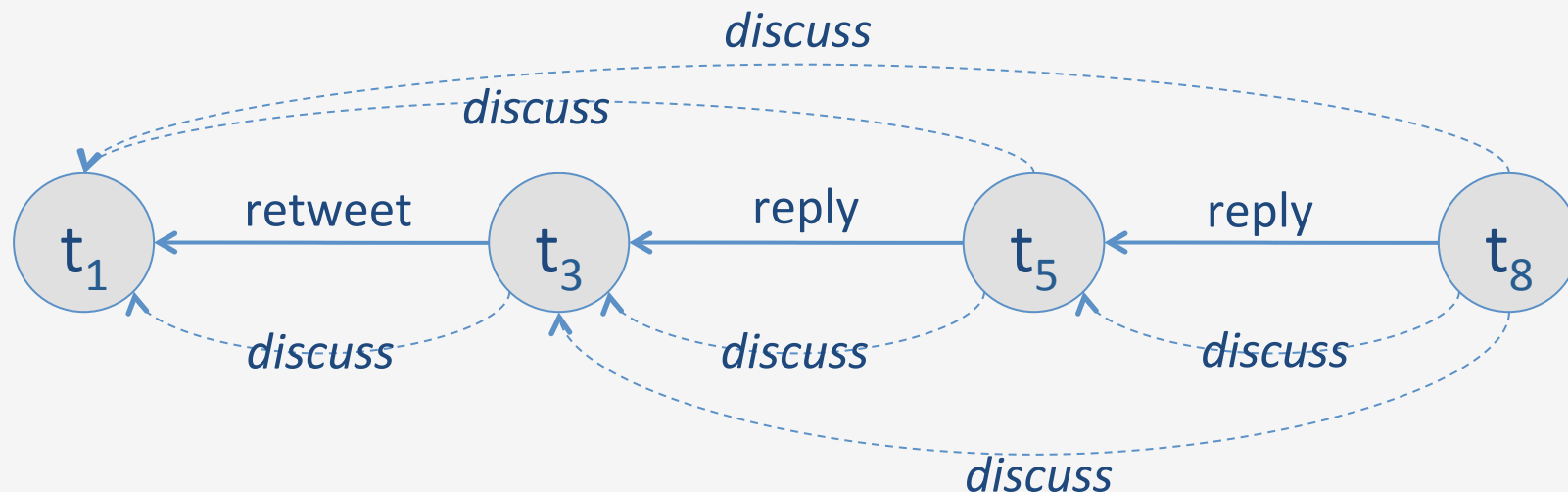


- The problem: **incrementing a materialization is easy, but decrementing a view is hard.**

- The impact of a tweet can be measured by checking how fast and how wide a tweet was discuss
- Twitter allows two traceable ways of discussing a tweet:
 - reply: a user reply to a tweet of another user (it always retweet the original tweet)
 - retweet: a user propagates to his/her followers an interesting tweet
- For example



- More formally
 - `:discuss` a `owl:TransitiveProperty` .
 - `:retweet` `rdfs:subPropertyOf` `:discuss` .
 - `:reply` `rdfs:subPropertyOf` `:discuss` .
- The full materialization (ignoring publication) time is straight forward to compute



- A query like the following one can be easily evaluated on the materialization

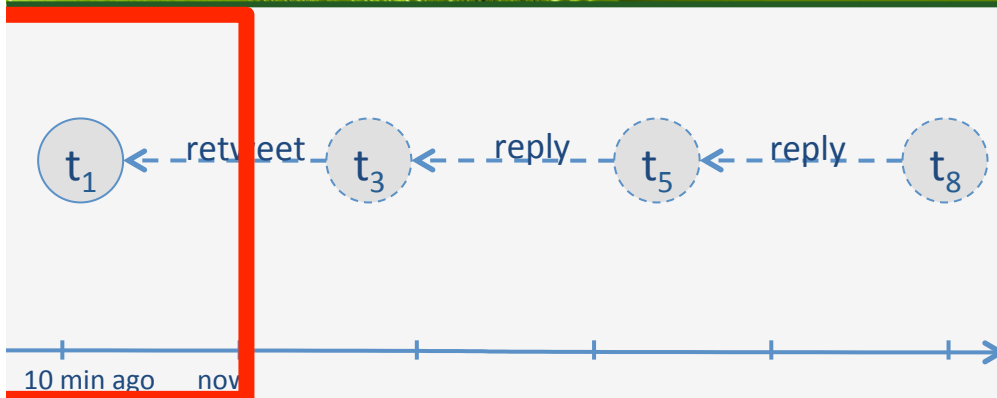
```
REGISTER STREAM OpinionSpreading COMPUTED EVERY 30s AS  
SELECT ?tweet (count(?tweetInTheDiscussion) AS ?impact  
FROM <http://www.streamreasoning.org/sr4ld2011/stream>  
WHERE {  
    ?tweet a sr:Tweet .  
    ?tweetInTheDiscussion sr:Tweet .  
    ?tweet sr:discuss ?tweetInTheDiscussion .  
}
```

- Results

?tweet	?impact
t1	3
t3	2
t5	1

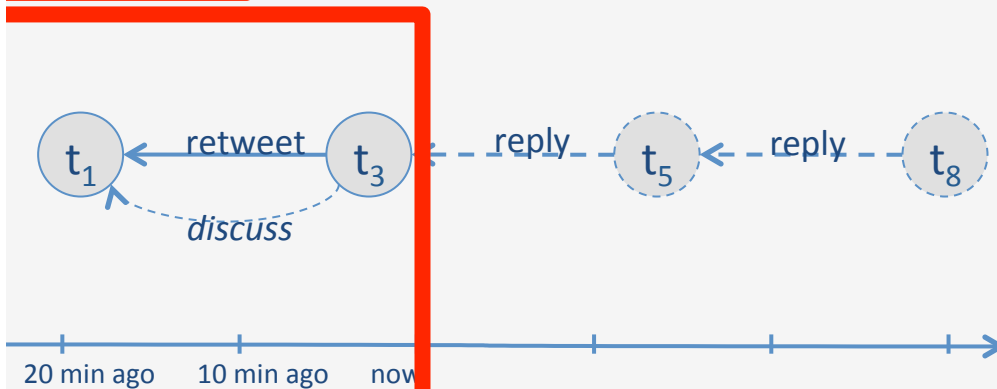
- Let's turn the previous SPARQL into a C-SPARQL one adding a window other the stream

```
REGISTER STREAM OpinionSpreading COMPUTED EVERY 30s AS
SELECT ?tweet (count(?tweetInTheDiscussion) AS ?impact
FROM STREAM <http://www.streamreasoning.org/sr4ld2011/stream>
[RANGE 40m STEP 10m]
WHERE {
    ?tweet a twd:Tweet .
    ?tweetInTheDiscussion twd:Tweet .
    ?tweet sr:discuss ?tweetInTheDiscussion
}
```



?tweet

?impact

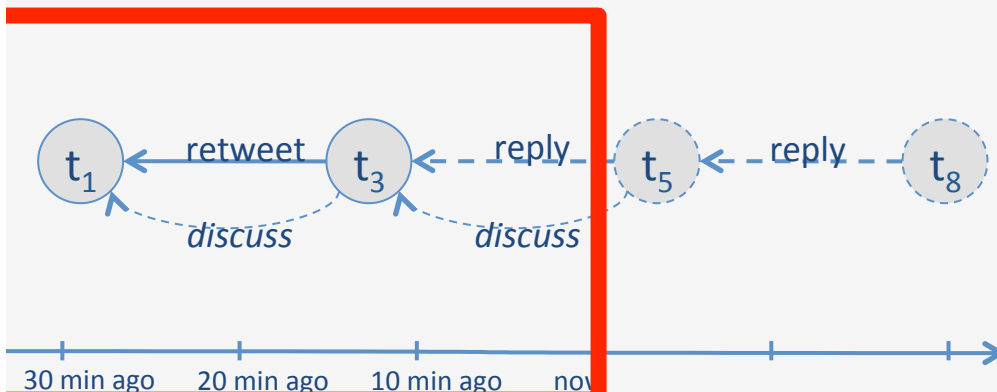


?tweet

?impact

t1

1



?tweet

?impact

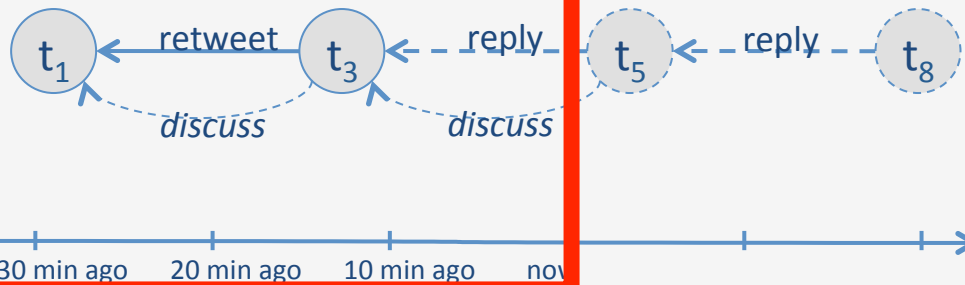
t1

1

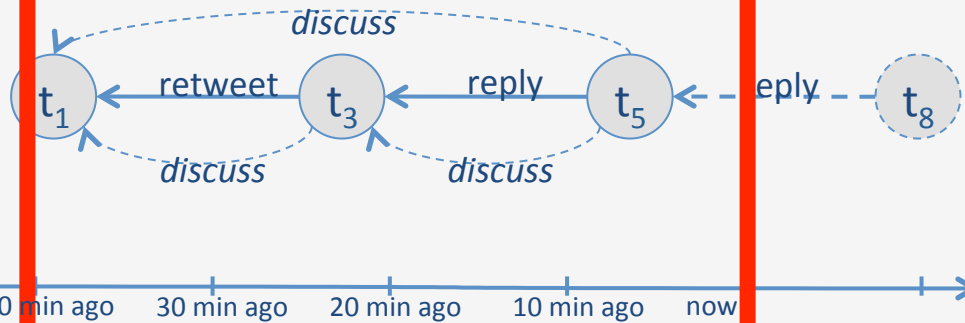
The Problem

An Example in Social Media – The results

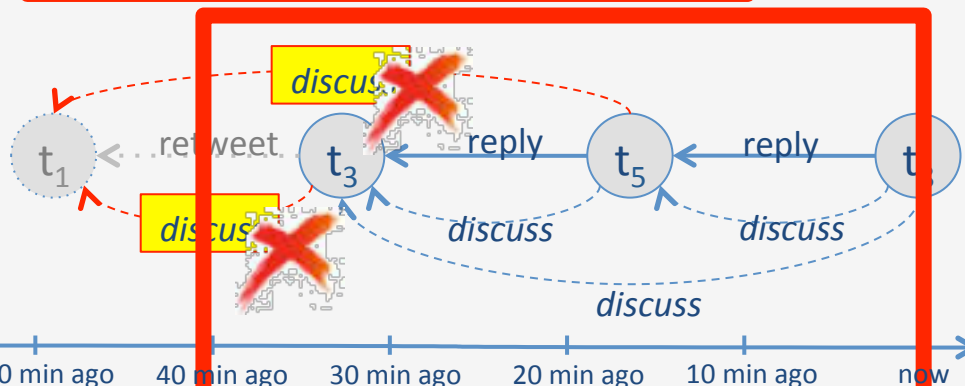
Heraklion
Greece
May 29th – June 2nd
2011



?tweet	?impact
t1	1



?tweet	?impact
t1	2
t3	1



?tweet	?impact
t3	2
t5	1

Stream Reasoning For Linked Data

E. Della Valle & J.Z. Pan

Heraklion
Greece
May 29th – June 2nd
2011



[3.2] Incremental Updates of RDFS++ Deductive Closures – State-of-the-Art

Emanuele Della Valle

emanuele.dellavalle@polimi.it

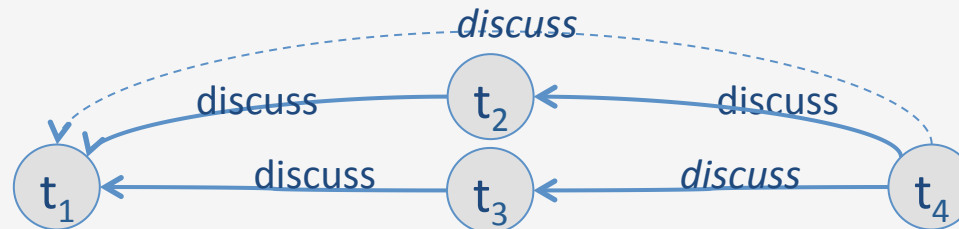
<http://emanueledellavalle.org>

- Introduction
- State-of-the-Art – the DRed Algorithm
 - Intuition
 - Expressing Ontology Languages as Rules
 - Declarative Version of Dred
 - Maintenance Predicates
 - Rewriting Functions
 - Example of Maintenance Program
 - Performances
- The Streaming Approach
- Conclusions

The State-of-the-Art approach is the DRed algorithm

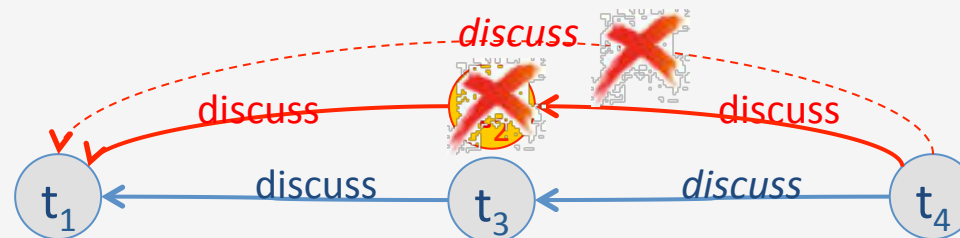
1. *Overestimation of deletion*: Overestimates deletions by computing all direct consequences of a deletion.
2. *Rederivation*: Prunes those estimated deletions for which alternative derivations (via some other facts in the program) exist.
3. *Insertion*: Adds the new derivations that are consequences of insertions to extensional predicates.

- Let's assume that we have the following materialized graph

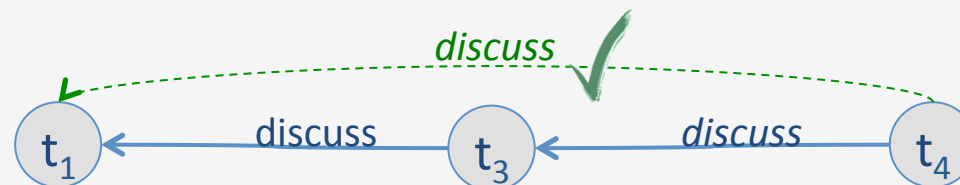


- While inserts are not problematic, deletion are difficult to handle. If we delete t_2 , we have

- overestimate the impact of the deletion** and mark for deletion $t_4 \rightarrow t_1$ that can be derived by $t_4 \rightarrow t_2$ and $t_2 \rightarrow t_1$



- look for alternative derivation** of $t_4 \rightarrow t_1$ and eventually find the chain $t_4 \rightarrow t_3$ and $t_3 \rightarrow t_1$



- RDF-S++ includes
 - `rdf:type`
 - `rdfs:subClassOf`
 - `rdfs:domain` and `rdfs:range`
 - `rdfs:subPropertyOf`
 - `owl:sameAs`
 - `owl:inverseOf`
 - `owl:TransitiveProperty`

- DRed works on ontologies express as rules
- Using rules is a best practice in implementing the logical entailment supported by ontology languages such as RDF-S and OWL2-RL.
- For example, the following production rules capture the semantics of

- `rdfs:subPropertyOf` and :

```
[prp-spo1:  
  (?p1, rdfs:subPropertyOf, ?p2), (?x, ?p1, ?y)  
  -> (?x, ?p2, ?y)  
]
```

- `owl:TransitiveProperty`

```
[prp-trp:  
  (?p, rdf:type, owl:TransitiveProperty),  
  (?x, ?p, ?y), (?y, ?p, ?z)  
  -> (?x, ?p, ?z)  
]
```

- In logic programming terminology, a set of rules can be seen as a logic program and an RDF graph can be stored in the extension of a single ternary predicate T.
- Under this assumption, the two rules above can be represented as follows:

```
prp - spol :  
  T(?x, ?p2, ?y) :-  
    T(?p1, rdfs:subPropertyOf, ?p2), T(?x, ?p1, ?y)
```

```
prp - trp :  
  T(?x, ?p, ?z) :-  
    T(?p, rdf:type, owl:TransitiveProperty),  
    T(?x, ?p, ?y), T(?y, ?p, ?z)
```

- A logic program is composed of a set of rules **R** that we can represent as $H :- B_1, \dots, B_n$
 - where H is the predicate that forms the head of the rule and
 - $B_1, \dots, B_n$ are the predicates that form the body of the rule.
- If we call the set of predicates in a logic program **P**, then we can formally assert that $H, B_i \in \mathbf{P}$.
- A **maintenance program**, which implements the declarative version of the DRed algorithm, can be **automatically derived** from the original program **with** a fixed set of **rewriting functions that uses** seven **maintenance predicates**

- Given a materialized predicate T and the set of extensional insertions T^{ins} to and deletions T^{del} from T^{before} , the goal of the rewriting functions is the definition of two maintenance predicates T^+ and T^-

Name	Content of the extension
T^{before}	the current materialization
T^{del}	the deletions
T^{ins}	the explicit insertions
T^{red}	the triples marked for deletion which can be alternatively re-derive
T^{after}	the materialization after the execution of the maintenance program
T^+	the net insertions required to maintain the materialization
T^-	the net deletions required to maintain the materialization

- We can divide the rewriting functions into two groups:
 - one applies to predicates,
 - one applies to rules.

- These rewriting functions use the maintenance predicates to introduce the rules that will store the materialization after the execution of the maintenance program in the extension of the predicate T^{after}

Predicate		
Name	Generator Parameter	Rewriting Result
δ_1^{after}	$P \in P$	$P^{after} \text{ :- } P^{before}, \text{ not } P^{del}$
δ_2^{after}	$P \in P$	$P^{after} \text{ :- } P^{pred}$
δ_3^{after}	$P \in P$	$P^{after} \text{ :- } P^{ins}$
δ^+	$P \in P$	$P^+ \text{ :- } P^{ins}, \text{ not } P^{before}$
δ^-	$P \in P$	$P^- \text{ :- } P^{del}, \text{ not } P^{ins}, \text{ not } P^{pred}$

- The rewriting functions for Predicates introduce the rules that populate the extensions of the predicates T^{del} , T^{red} , and T^{ins} .
- These three rewriting functions are executed for each rule that has the predicate T as head.

Rule		
Name	Generator Parameter	Rewriting Result
δ^{red}	$H :- B_1, \dots, B_n$	$H^{\text{red}} :- H^{\text{del}}, B_1^{\text{after}}, \dots, B_n^{\text{after}}$
δ^{del}	$H :- B_1, \dots, B_n$	$H^{\text{del}} :- B_1^{\text{before}}, \dots, B_{i-1}^{\text{before}}, B_i^{\text{del}}, B_{i+1}^{\text{before}}, \dots, B_n^{\text{before}}$
δ^{ins}	$H :- B_1, \dots, B_n$	$H^{\text{ins}} :- B_1^{\text{after}}, \dots, B_{i-1}^{\text{after}}, B_i^{\text{ins}}, B_{i+1}^{\text{after}}, \dots, B_n^{\text{after}}$

- Note that
 - δ^{red} rewrites each rule in exactly one maintenance rule,
 - δ^{del} and δ^{ins} rewrite each rule with n atoms in the body B_i into n maintenance rules.

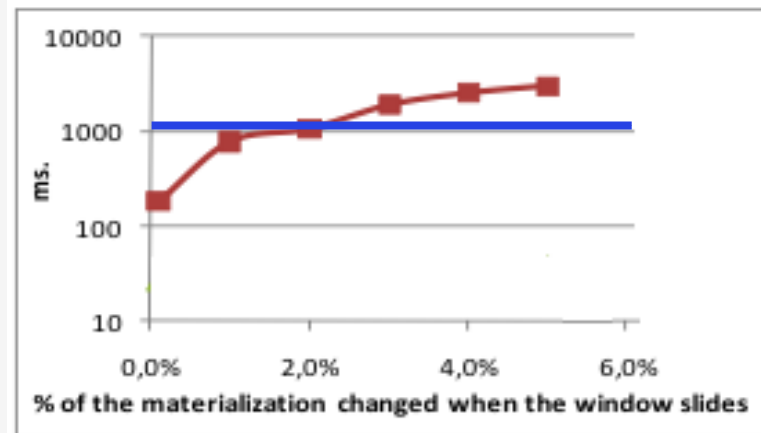
- Consider the rule
- R:** $T(?x, ?p, ?z) :-$
 $T(?p, \text{rdf:type}, \text{owl:TransitiveProperty}),$
 $T(?x, ?p, ?y), T(?y, ?p, ?z)$
- By applying the rewriting functions we obtain the maintenance program

$T^{after}(\text{?x}, \text{?p}, \text{?y})$	$:- T^{before}(\text{?x}, \text{?p}, \text{?y}), \text{ not } T^{del}(\text{?x}, \text{?p}, \text{?y})$	$\delta_1^{after}(T)$
$T^{after}(\text{?x}, \text{?p}, \text{?y})$	$:- T^{red}(\text{?x}, \text{?p}, \text{?y})$	$\delta_2^{after}(T)$
$T^{after}(\text{?x}, \text{?p}, \text{?y})$	$:- T^{ins}(\text{?x}, \text{?p}, \text{?y})$	$\delta_3^{after}(T)$
$T^+(\text{?x}, \text{?p}, \text{?y})$	$:- T^{ins}(\text{?x}, \text{?p}, \text{?y}), \text{ not } T^{before}(\text{?x}, \text{?p}, \text{?y})$	$\delta^+(T)$
$T^-(\text{?x}, \text{?p}, \text{?y})$	$:- T^{del}(\text{?x}, \text{?p}, \text{?y}), \text{ not } T^{ins}(\text{?x}, \text{?p}, \text{?y}),$ $\text{ not } T^{red}(\text{?x}, \text{?p}, \text{?y})$	$\delta^-(T)$

[...]

$T^{red}(\text{?x}, \text{?p}, \text{?z})$	$:-$	$T^{del}(\text{?x}, \text{?p}, \text{?z}), T^{after}(\text{?x}, \text{?p}, \text{?y}),$ $T^{after}(\text{?y}, \text{?p}, \text{?z}),$ $T^{after}(\text{?p}, \text{rdf:type}, \text{owl:TransitiveProperty})$	$\delta^{red}(T)$
$T^{del}(\text{?x}, \text{?p}, \text{?z})$	$:-$	$T^{del}(\text{?p}, \text{rdf:type}, \text{owl:TransitiveProperty}),$ $T^{before}(\text{?x}, \text{?p}, \text{?y}), T^{before}(\text{?y}, \text{?p}, \text{?z})$	$\delta^{del}(T)$
$T^{del}(\text{?x}, \text{?p}, \text{?z})$	$:-$	$T^{before}(\text{?p}, \text{rdf:type}, \text{owl:TransitiveProperty}),$ $T^{del}(\text{?x}, \text{?p}, \text{?y}), T^{before}(\text{?y}, \text{?p}, \text{?z})$	$\delta^{del}(T)$
$T^{del}(\text{?x}, \text{?p}, \text{?z})$	$:-$	$T^{before}(\text{?p}, \text{rdf:type}, \text{owl:TransitiveProperty}),$ $T^{before}(\text{?x}, \text{?p}, \text{?y}), T^{del}(\text{?y}, \text{?p}, \text{?z})$	$\delta^{del}(T)$
$T^{ins}(\text{?x}, \text{?p}, \text{?z})$	$:-$	$T^{after}(\text{?p}, \text{rdf:type}, \text{owl:TransitiveProperty}),$ $T^{ins}(\text{?x}, \text{?p}, \text{?y}), T^{after}(\text{?y}, \text{?p}, \text{?z})$	$\delta^{ins}(T)$
$T^{ins}(\text{?x}, \text{?p}, \text{?z})$	$:-$	$T^{after}(\text{?p}, \text{rdf:type}, \text{owl:TransitiveProperty}),$ $T^{after}(\text{?x}, \text{?p}, \text{?y}), T^{ins}(\text{?y}, \text{?p}, \text{?z})$	$\delta^{ins}(T)$
$T^{ins}(\text{?x}, \text{?p}, \text{?z})$	$:-$	$T^{ins}(\text{?p}, \text{rdf:type}, \text{owl:TransitiveProperty}),$ $T^{after}(\text{?x}, \text{?p}, \text{?y}), T^{after}(\text{?y}, \text{?p}, \text{?z})$	$\delta^{ins}(T)$

- The maintenance time grows as a function of the % of changes (see red line in the graph below).
- In a knowledge base mostly consisting in facts that relates instances with transitive properties (as in our running example) if the changes interest more that 2% of the knowledge base it no longer pays off to incrementally maintain the materialization.



- Computing the entire materialization from scratch takes less (see blue line in the graph above).

Stream Reasoning For Linked Data

E. Della Valle & J.Z. Pan

Heraklion
Greece
May 29th – June 2nd
2011



[3.3] Incremental Updates of Deductive Closures – Streaming Approach

Emanuele Della Valle

emanuele.dellavalle@polimi.it

<http://emanueledellavalle.org>

- Introduction
- State-of-the-Art – the DRed Algorithm
- The Streaming Approach
 - Intuition
 - Formalization
 - Maintenance Predicates
 - Rewriting Functions
 - Efficient Deletion
 - Handling Multiple Derivations
 - Example of Maintenance Program
 - Performances
- Conclusions

- Assumption
 - Insertions and deletions are triples respectively entering and exiting the window
 - The window size is known
- Therefore
 - The time when each triple will expire is known and determined by the window size
 - E.g. if the window is 10s long a triple entering at time t will exit at time $t+10s$
 - Note: all knowledge can be annotated with an expiration time
 - i.e., background knowledge is annotated with $+\infty$

The Streaming algorithm

1. deletes all triples (asserted or inferred) that have just expired
2. computes the entailments derived by the inserts,
3. annotates each entailed triple with a expiration time, and
4. eliminates from the current state all copies of derived triples except the one with the highest timestamp.

The Streaming Approach Intuition

Heraklion
Greece
May 29th – June 2nd
2011



TS	Triples in the Window	Entailments in the Window
1	$A \xrightarrow{[11]} B$	
2	$A \xrightarrow{[11]} B \xrightarrow{[12]} C$	
3	$A \xrightarrow{[11]} B \xrightarrow{[12]} C \xrightarrow{[13]} D$	
4	$A \xrightarrow{[14]} E \xrightarrow{[14]} D$ $A \xrightarrow{[11]} B \xrightarrow{[12]} C \xrightarrow{[13]} D$	
11	$A \xrightarrow{[14]} E \xrightarrow{[14]} D$ $A \xrightarrow{[11]} B \xrightarrow{[12]} C \xrightarrow{[13]} D$	
12	$A \xrightarrow{[14]} E \xrightarrow{[14]} D$ $A \xrightarrow{[12]} B \xrightarrow{[13]} C \xrightarrow{[13]} D$	
13	$A \xrightarrow{[14]} E \xrightarrow{[14]} D$ $A \xrightarrow{[13]} C \xrightarrow{[13]} D$	

Multiple Derivation

If a longer lasting derivation of a triple already entailed is found the expiration time of the entailed triple is updated

Efficient Deletion

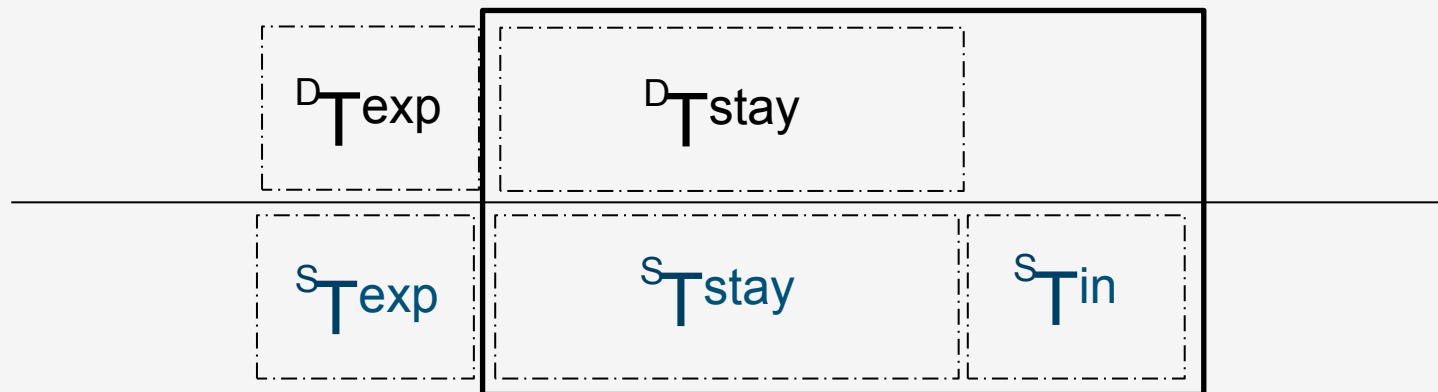
Deletion is achieved by lookups of triples in an hashmap. No over deletion occurs

- At the core of our approach there is the notion of **expiration time** for any transient piece of knowledge
- The interpretation of the expiration time is simply the time at which a piece of knowledge will not be valid anymore
- A triple t annotated with an expiration time e is denoted as $t[e]$
- Definitions
 - A triple $\mathbf{t_1[e_1]}$ is said to be **longer-lasting than** a triple $\mathbf{t_0[e_0]}$ iff its expiration time is greater, i.e., $e_1 > e_0$.
 - A triple $\mathbf{t_1[e_1]}$ is said to be **a duplicate of** a triple $\mathbf{t_2[e_2]}$ iff they have the same subject, predicate and object, but different expiration times (i.e., $t_1 = t_2$ and $e_1 \neq e_2$).
 - A triple $\mathbf{t_1}$ is said to be **renewed by** another triple $\mathbf{t_2}$, iff t_2 is longer-lasting than t_1 and they are duplicates.

- ST^{in} : the predicate whose extension contains the triples in the stream that enter the window during the transition from τ_0 to τ_1 .
- ST^{stay} : the predicate whose extension contains the triples in the stream that stay in the window before and after the transition from τ_0 to τ_1 .
- ST^{exp} : the predicate whose extension contains the triples in the stream that exit the window during the transition from τ_0 to τ_1 , and therefore expire.



- $D\mathcal{T}^{\text{exp}}$: the predicate whose extension contains the derived triples that expire during the transition from τ_0 to τ_1 .
- $D\mathcal{T}^{\text{stay}}$: the predicate whose extension contains the derived triples that stay in the window before and after the transition from τ_0 to τ_1 .



- DT^{new} : the new triples whose derivation is triggered by the triples in ST^{in} and are not already in DT^{stay} .
- DT^{renew} : the triples whose derivation is triggered by triples in ST^{in} and happen to renew triples already in DT^{stay} .
- DT^{obs} : the triples that were renewed by triples in DT^{renew} , thus becoming obsolete.
- $wipT = ST^{in} \cup ST^{stay} \cup DT^{stay} \cup DT^{new} \cup DT^{renew}$ (work in progress): it denotes the overall knowledge that can trigger new derivations
- T^+ : the triples to be added to the materialization to complete the transition.
- T^- : the triples to be deleted from the materialization to complete the transition.

- Note that by definition the expiration time of the triples in $S_{t^{exp}} \cup D_{t^{exp}}$ is less than $now()$.
 - $S_{t^{exp}}$ contains streaming triples “physically” expiring and
 - $D_{t^{exp}}$ contains derived triples that expire because one of their premises for derivation expires, thus invalidating the entailment.
- If a hasmap is used to access all triples in T by their expiration time, retracting $S_{t^{exp}} \cup D_{t^{exp}}$ is efficient, i.e., it only requires lookups in a hashmap.
- We found a way to bypass the bottleneck of DRed algorithm!

- As for the [Voltz2006] approach given a deductive rule

$$\mathbf{r}: H :- B_1, \dots, B_n$$

- Maintenance rules can be automatically generated using the two following rewriting functions

- New** production rules $\mathcal{R}^{new}$: one rule for each i , $1 \leq i \leq n$

$$\mathbf{r}_i^{new}: {}^D H^{new}[e] \leftarrow {}^{wip} B_1[e_1], \dots, {}^S B_i^{in}[e_i], \dots, {}^{wip} B_n[e_n], \text{ not } {}^D H^{stay},$$

$$e = \min(e_1, \dots, e_n)$$

- Renew** production rules $\mathcal{R}^{renew}$: one rule for each i , $1 \leq i \leq n$

$$\mathbf{r}_i^{renew}: {}^D H^{renew}[e], {}^D H^{obs}[e_{old}] \leftarrow {}^{wip} B_1[e_1], \dots, {}^D B_i^{new}[e_i], \dots, {}^{wip} B_n[e_n],$$

$${}^{wip} H[e_{old}],$$

$$e > e_{old}, e = \min(e_1, \dots, e_n)$$

- we observe that rules R^{new} derive triples that are certainly new.
- Note that these rules can recursively fire one another, because they produce triples in DT^{new} .
 - The “newness” of the produced triples is guaranteed by the presence of *not* DH^{stay} in the rule bodies.
 - This, together with the assumption that windows contain no duplicates, prevents the program from re-deriving duplicate triples that are obsolete w.r.t. those already derived.
- Also note that during the execution of r^{new} only DT^{new} grows, among the component of $wipT$, and does so up to its fixpoint, whereas DT^{renew} remains empty.

- we observe that rules r^{renew} check if any of the already derived triples should be renewed by a longer-lasting derived duplicate.
- If this is the case, they put the obsolete duplicate in DH^{obs} and the “newer” duplicate in DH^{renew} (the two actions corresponding to the two predicates in the rule’s head).
- As for the previous cases, these rules can recursively fire one another because any new triple they add to DT^{renew} also belongs to $wipT$.

- Also note that in this case multiple derivations of the same truth are possible, thus leading to the generation of duplicates, and also possibly to the re-derivation of clones.
- Therefore, triples already renewed may be renewed again, and clones may belong to both DT^{renew} and DT^{obs} . However, the condition on the minimum e guarantees that these re-derivations do not trigger the whole re-materialization of the entailments of the clones.
- Last, we observe that during the second phase of rule execution, r^{new} only affects DT^{renew} and DT^{obs} among the component of $wipT$.

- $DT^- = DT^{\text{exp}} \cup DT^{\text{obs}}$
 - $DT^+ = DT^{\text{new}} \cup DT^{\text{renew}} \setminus DT^{\text{obs}}$
 - $T^- = ST^{\text{exp}} \cup DT^-$
 - $T^+ = ST^{\text{in}} \cup DT^+$
-
- T^- causes the deletion of all triples that expired and all those that became obsolete
 - T^+ adds all triples that entered the window, those newly derived, and the ones “finally” renewed.
 - By “finally” we mean that only the longest-lasting duplicate of each re-derived triple is retained, due to the set-subtraction of DT^{obs} from DT^{renew} .

- Consider the rule

R: $T(?x, ?p, ?z) :-$
 $T(?p, \text{rdf:type}, \text{owl:TransitiveProperty}),$
 $T(?x, ?p, ?y), T(?y, ?p, ?z)$

- By applying the production rules r^{new} we obtain

$$DT^{\text{new}}(?x, ?p, ?z)[e] \leftarrow ST^{\text{in}}(?p, \text{rdf:type}, \text{owl:TransitiveProperty})[e_1],$$

$$wipT(?x, ?p, ?y)[e_2], wipT(?y, ?p, ?z)[e_3],$$

$$\text{not } DT^{\text{stay}}(?x, ?p, ?z)[e], e = \min(e_1, e_2, e_3)$$

$$DT^{\text{new}}(?x, ?p, ?z)[e] \leftarrow wipT(?p, \text{rdf:type}, \text{owl:TransitiveProperty})[e_1],$$

$$ST^{\text{in}}(?x, ?p, ?y)[e_2], wipT(?y, ?p, ?z)[e_3],$$

$$\text{not } DT^{\text{stay}}(?x, ?p, ?z)[e], e = \min(e_1, e_2, e_3)$$

$$DT^{\text{new}}(?x, ?p, ?z)[e] \leftarrow wipT(?p, \text{rdf:type}, \text{owl:TransitiveProperty})[e_1],$$

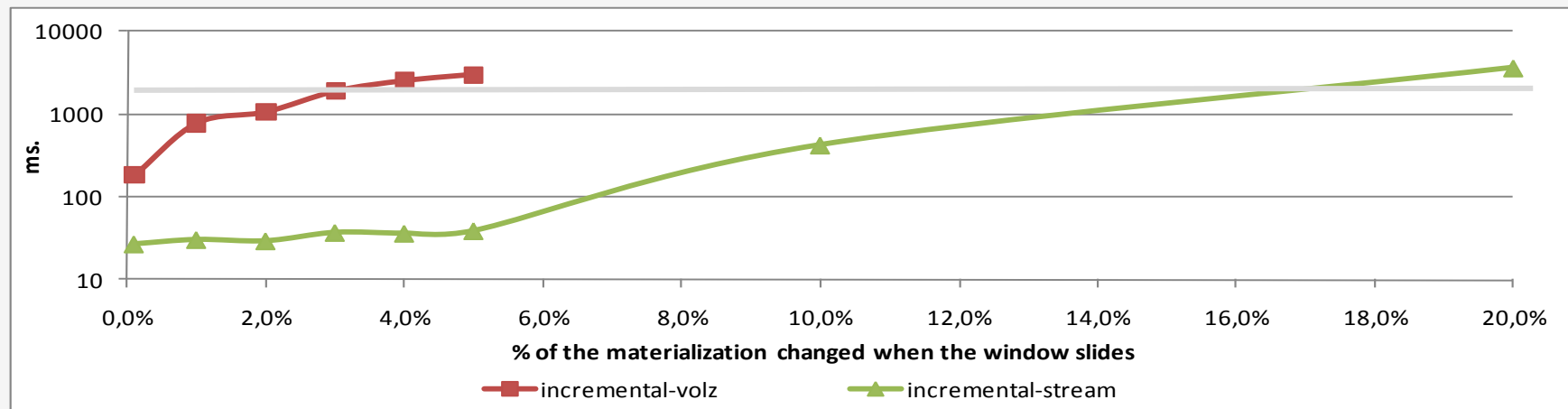
$$wipT(?x, ?p, ?y)[e_2], ST^{\text{in}}(?y, ?p, ?z)[e_3],$$

$$\text{not } DT^{\text{stay}}(?x, ?p, ?z)[e], e = \min(e_1, e_2, e_3)$$

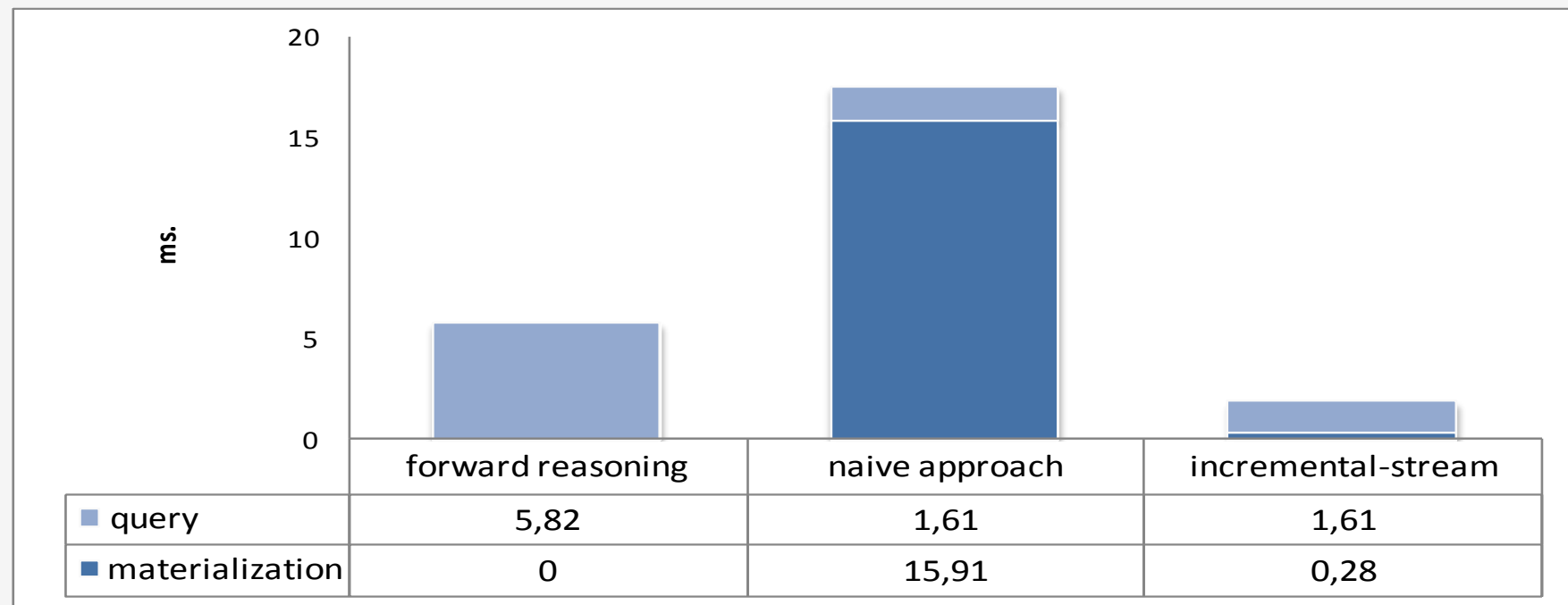
- By applying the production rules r^{renew} we obtain

$$\begin{aligned}
 DT^{\text{renew}}(?x, ?p, ?z)[e], DT^{\text{renewed}}(?x, ?p, ?z)[e_{\text{old}}] &\leftarrow DT^{\text{new}}(?p, \text{rdf:type}, \text{owl:TransitiveProperty})[e_1], \\
 &\quad wipT(?x, ?p, ?y)[e_2], wipT(?y, ?p, ?z)[e_3], \\
 &\quad e > e_{\text{old}}, e = \min(e_1, e_2, e_3) \\
 DT^{\text{renew}}(?x, ?p, ?z)[e], DT^{\text{renewed}}(?x, ?p, ?z)[e_{\text{old}}] &\leftarrow wipT(?p, \text{rdf:type}, \text{owl:TransitiveProperty})[e_1], \\
 &\quad DT^{\text{new}}(?x, ?p, ?y)[e_2], wipT(?y, ?p, ?z)[e_3], \\
 &\quad e > e_{\text{old}}, e = \min(e_1, e_2, e_3) \\
 DT^{\text{renew}}(?x, ?p, ?z)[e], DT^{\text{renewed}}(?x, ?p, ?z)[e_{\text{old}}] &\leftarrow wipT(?p, \text{rdf:type}, \text{owl:TransitiveProperty})[e_1], \\
 &\quad wipT(?x, ?p, ?y)[e_2], DT^{\text{new}}(?y, ?p, ?z)[e_3], \\
 &\quad e > e_{\text{old}}, e = \min(e_1, e_2, e_3)
 \end{aligned}$$

- Hypothesis
 - Background knowledge do not change and it is fully materialized
 - Changes only take place in the window
- An experiment comparing the time required to compute a new materialization using
 - Re-computing from scratch (i.e., 1250 ms in our setting)
 - State of the art incremental approach [Volz, 2005]
 - Our approach
- Results at increasing % of the materialization changed when the window slides



- Comparison of the average time needed to answer a C-SPARQL query using
 - a forward reasoner,
 - the naive approach of re-computing the materialization
 - our approach



[ESWC2010] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, Michael Grossniklaus. Incremental Reasoning on Streams and Rich Background Knowledge. In. 7th Extended Semantic Web Conference (ESWC 2010)

[Ceri1994] Stefano Ceri, Jennifer Widom: Deriving Incremental Production Rules for Deductive Data. Inf. Syst. 19(6): 467-490 (1994)

[Volz2005] Raphael Volz, Steffen Staab, Boris Motik: Incrementally Maintaining Materializations of Ontologies Stored in Logic Databases. J. Data Semantics 2: 1-34 (2005)

Stream Reasoning For Linked Data

E. Della Valle & J.Z. Pan

Heraklion
Greece
May 29th – June 2nd
2011



[3.4] Conclusions

Emanuele Della Valle

emanuele.dellavalle@polimi.it

<http://emanueledellavalle.org>

- **RDF Streams**
 - Notion defined
- **C-SPARQL**
 - Syntax and semantics defined as a SPARQL extension
 - Engine designed
 - Engine implemented based on the decision to keep stream management and query evaluation separated
- **Experiments with C-SPARQL** under simple RDF entailment regimes
 - window based selection of C-SPARQL outperforms the standard FILTER based selection
 - having formally defined C-SPARQL semantics algebraic optimizations are possible
- **Experiment with C-SPARQL under RDFS++ entailment regimes**
 - efficient incremental updates of deductive closures investigated
 - our approach outperform state-of-the-art when updates comes as stream

- Relation with data-stream systems
 - Notion of RDF stream :-|
- Query languages for semantic streams
 - C-SPARQL :-D
- Reasoning on Streams
 - Formal representations for stream reasoning
 - :-P
 - Notions of soundness and completeness
 - :-P
 - Efficient incremental updates of deductive closures
 - ESWC 2010 paper :-) ... but other approaches can be proposed
(see next session!)
 - How to combine streams and background knowledge
 - ESWC 2010 paper :-| ... but a lot needs to be studied ...
- Dealing with incomplete & noisy data
 - :-P
- Distributed and parallel processing
 - :-P

Stream Reasoning For Linked Data

E. Della Valle & J.Z. Pan

Heraklion
Greece
May 29th – June 2nd
2011



[3] Stream Reasoning techniques for RDFS++

Emanuele Della Valle

emanuele.dellavalle@polimi.it

<http://emanueledellavalle.org>