Inferencing and Truth Maintenance in RDF Schema

Exploring a naive practical approach

Jeen Broekstra
jeen.broekstra@aidministrator.nl

Arjohn Kampman
arjohn.kampman@aidministrator.nl

Abstract Contrary to earlier reports in literature, exhaustive forward inferencing is a feasible approach for practical RDF. It is sufficiently fast and the increase in necessary storage size is sufficiently small to make it work. Benefits of this approach are low-cost design and implementation, and very cheap query answering, since this task is reduced to simple lookup without inferencing.

A potential problem of exhaustive forward inferencing is how to deal with statement deletion (an aspect often ignored thus far): when a statement is removed, some of its consequences may also have to be removed. The naive approach is to simply recalculate the entire deductive closure of the RDF store. A more sophisticated approach is based on truth maintenance: it tracks all deductive dependencies between statements, and uses this to determine which other statements will have to be removed as a consequence of a single deletion. This approach has the additional advantage of having deductive dependencies available for other tasks, such as business logic and change tracking.

We give a detailed algorithm for such truth maintenance for RDF(S), and we compare the performance of this algorithm with that of the naive recomputation approach.

1 Introduction

The Resource Description Framework (RDF) [7] specifies a simple model for knowledge representation. RDF Schema (RDFS) [1] adds additional expressive power and semantics to this basic model. The combined language has a simple propositional logic as its foundation, the semantics of which are described in the RDF Model Theory (MT) [5].

We have studied a number of practical issues that arise when computing the deductive closure of an RDF Schema and dealing with the consequences of delete operations. First, we present a simple, scalable approach to RDF MT inferencing in an exhaustive forward-chaining fashion. Second, we present a Truth Maintenance algorithm that makes use of dependencies between statements to deal with 'non-monotonous' updates (i.e. delete operations) to an RDF Schema knowledge base. We present how this algorithm has been implemented in the Sesame [2] architecture, and present the results of several benchmark tests we have undertaken with our approach.

The purpose of dependency tracking between statements as presented in this paper is twofold. First, the information about the dependency relations between statements is necessary for several high-level services on RDF repositories, such as change tracking and statement-level security (see [3] for details). Second, the goal is to achieve a performance improvement in removal operations when compared to brute-force approaches.

2 Related Work

There is a large body of literature available on efficient algorithms for transitive closure computing. For example, in [10] a thorough analysis of the problem is given and a set of algorithms for maintaining transitive closures and transitive reductions of digraphs is presented. While such closure algorithms are highly relevant for the problem at hand, we do not consider this approach in this paper: our aim is not to present a highly efficient algorithm for inferencing, but to implement a naive exhaustive forward chaining approach (which is very low-cost to design and implement) and ”see how far we can stretch it”. Claims in earlier literature have often dismissed the idea of such a naive approach out of hand, but we feel it is worthwile to explore the idea, to determine empirically how scalable and practical such an approach can be.

In [4], Doyle proposes a Truth Maintenance System, to allow reasoning programs to make assumptions and subsequently revise their beliefs when discoveries contradict these assumptions.

The system proposed in this paper is much simpler than the system by Doyle: it only has to deal with propagation of disbelief (i.e. the TMS is only invoked when propositions are retracted). In fact, the algorithm that we propose is a drastically simplified and altered version of the justification-based TMS proposed in [4].

3 Inferencing in RDF

3.1 The RDF Model Theory

The RDF Model Theory [5] is a specification of a model-theoretic semantics for RDF and RDF Schema, and some basic results on entailment. Of particular interest to us is that it presents a set of entailment rules for computing the deductive closure of an RDF Schema Model. These entailment rules are at the core of why a TMS is needed in a system that interprets RDF Schema semantics.

The entailment rules infer new RDF statements based on the presence of other statements. However, the RDF MT also asserts a set of RDF statements which are considered true regardless of what other information is present. These axiom statements deal exclusively with the semantics of the RDF(S) primitives themselves, and assert such truths as ”subPropertyOf is a Property”.

The claim of [5] is that the deductive closure of an RDF Schema is computed by recursively applying these entailment rules to the set of RDF statements.

3.2 Forward chaining closure computing

In [8], Ora Lassila notes that using brute-force, exhaustive, iterative application of the RDF MT entailment rules is not a realistic option:

	triggers rule:												
rule:	1	2	3	4a	4b	5a	5b	6	7a	7b	8	9	10	11
1		*	*	*			*	*				*		
2		*	*				*	*	*	*		*	*	*
3		*	*				*	*	*	*		*	*	*
4a		*	*					*				*		
4b		*	*					*				*		
5a						*		*						
5b		*	*					*						
6		*	*			*	*	*	*	*	*	*	*	*
7a		*	*					*			*	*		
7b		*	*					*						
8								*			*	*		
9			*				*	*	*	*		*	*	*
10		*	*		*	*		*						
11		*	*					*			*	*		

Table 1: Dependencies between RDFS entailment rules

> The rules are highly redundant and their brute-force, exhaustive, iterative application is not a realistic way of computing the closure. For example, give a graph with only one triple, the rules [...] would generate 17 new triples, but would also result in 493 attempts to add a redundant triple. [.....] another issue is that the application of the rules may result in the addition of a large number of new triples in the database [...].

However, as we will show in this section, with a few optimizations it turns out to be quite feasible to apply the RDF MT entailment rules in a exhaustive forward chaining strategy, and our experiments show that on RDF datasets taken from practice, the addition of these new triples does not result in problematic sizes of the repository.

Consider table 1, which shows the dependency relations between the entailment rules. A rule $\phi_1 \rightarrow \psi_1$ is dependent on another rule $\phi_2 \rightarrow \psi_2$ if there is some conclusion $s \in \psi_1$ that matches a premise $p \in \phi_2$. A number of things become apparent immediately from table 1: first, rule 1 is not dependent on any other rules. This means that for one run of the closure-computing mechanism this rule only needs to be invoked once. Second, rule 6 seems to be computationally heavy, as it is triggered by every other rule. However, if we examine rule 6 we see that it has to do with subProperty relations, which is a feature that is comparatively rarely used.

Sesame uses a forward chaining algorithm to compute and store the closure during any transaction that adds data to the repository. The algorithm runs iteratively over the entailment rules, but makes use of the dependencies between entailment rules shown above to eliminate most redundant inferencing steps. Specifically, in any iteration n of the algorithm only those entailment rules are invoked which have a dependency on a rule that produced new statements in iteration $n - 1$.

This exhaustive forward chaining approach, although by many considered to be non-scalable, turns out to perform surprisingly well. We used a number of different

data sets in testing this approach. The selected data sets are not toy examples, but actual sets that are in use in various projects, of different sizes and complexity, and we believe this selection to give an interesting cross-section of different use cases for RDF.

- **OpenWine** is an open source data set that contains information about different wines. It is available from `http://www.openwine.org/`.
- **SUO** stands for "Standard Upper Ontology" and is a DAML+OIL representation of the SUO IEEE effort to standardize an upper ontology[1]. The DAML+OIL file is available from `http://www.daml.org/ontologies/uri.html`.
- **CIA** is an RDF representation of the CIA World Factbook. It is an enhanced version of the RDF representation produced by the On-To-Knowledge IST project[2] and is available from `http://sesame.aidministrator.nl/testsets/CIA/`.
- **Wordnet** is a data set containing the Wordnet 1.6 schema (`wordnet-20000620.rdfs`) and the set of nouns (`wordnet_nouns-20010201.rdf`). These files are available for download at `http://www.semanticweb.org/library`.

As can be seen, the increase in number of statements through closure computing varies per data set, however in most cases it is well under control and does not exceed 50%. The only exceptions to this rule is the SUO data set. This large increase is caused by the fact that it consists exclusively of a large class hierarchy that is both broad and deep: since virtually every statement in the set is a schema statement a lot of inferencing rules are applicable to all statements.

3.3 Backward Chaining closure computing

An alternative approach towards computing the RDF MT closure of an RDF model employs a backward chaining strategy, which combines best with an on-demand/just-in-time approach towards closure computing (i.e. instead of computing the entire closure in advance, only the relevant part of the closure is computed whenever a particular query is being resolved). The main advantages of such an approach are the decrease in required storage size and upload time, while the main disadvantage is the decrease in performance of query answering. In our current setup, we have chosen to give query performance a high priority and therefore have chosen to explore the possibilities of forward chaining first. However, it is recognized that when the complexity of the model theory and the expressivity of the modeling language increase (for example when moving from RDF Schema to OWL Lite [11]), the disadvantage of larger storage space may at some point outweigh the advantage of faster querying, and an adjustment in strategy will need to be made. The remainder of this paper further explores the performance and the limits of the forward chaining strategy, and the consequences this strategy has for other operations, such as removal.

[1] See `http://suo.ieee.org/`
[2] see `http://www.ontoknowledge.org/`

4 Truth Maintenance

As we have seen in section 2, truth maintenance in RDF models is a comparitively simple problem. Since RDF is monotonic in nature, truth maintenance only becomes important when a statement is retracted. In other words, we are dealing with a single aspect of truth maintenance: *disbelief propagation*[9].

Our approach is further simplified by a practical assumption that is being made: only explicit statements can be retracted. The current algorithm does not deal with the belief revision involved in retracting derived statements, although it could be extended to cope with this scenario as well.

4.1 A brute-force approach

In our setting, where the complete closure is stored by means of a forward chaining inferencer, truth maintenance involves 'physical' retraction of statements that are no longer justified. A brute-force algorithm involves, quite simply, discarding all statements that were inferred and re-computing the closure.

The obvious benefit of the brute-force approach is that no additional bookkeeping is necessary, apart from whether a statement is explicit or derived.

A downside to this approach is that it makes even simple delete operations computationally expensive (see table 8 for comparitive results).

4.2 A justification-based TM algorithm for RDF

In this section, we present our justification-based Truth Maintenance algorithm. The algorithm makes use of the dependency relations between entailment rules (see table 1). More specifically, it tracks, for each statement in the model, of which other statements it is dependent, or put another way, which other statements *justify* it (cf. [4]).

As we have mentioned earlier, the purpose of dependency tracking is twofold: the metadata thus acquired is necessary for services such as repository change tracking and statement-level security policies (as described in [6]), and furthermore the aim is to achieve an improvement in the performance of removal operations when compared to brute-force approaches.

The algorithm will be executed at the end of any update transaction $\mathcal{T}$ which contains one or more delete operations.

We call B the set of believed facts. A fact $f \in B$ is labeled $explicit$ if it is asserted explicitly. If it is not explicitly asserted, f is labeled $derived$.

S is the set of justifications. Each justification $s \in S$ is of the form $\langle f_s, d1_s, d2_s \rangle$: $f_s, d1_s, d2_s \in B$. f_s is the fact justified by s. $d1_s$ and $d2_s$ are justifying facts for f_s, where $d1_s$ corresponds to the first premise of the entailment rule that produced s, and $d2_s$ to the second premise (note that the entailment rules always have at most two premises). When f_s is an RDF MT axiom statement, $d1_s = \alpha$. When s is justification

```
01. for each f ∈ D:
02.   if (f = explicit) then
03.      label f derived;
04.      remove s⟨f, ∅, ∅⟩ from S
05.   else
06.      remove f from D;
07.   endif;
08. end for;
09. repeat
10.   let removed := false;
11.   for each f ∈ D:
12.      if (∀s⟨f_s, d1_s, d2_s⟩ ∈ S : f_s ≠ f) then
13.         remove f from B;
14.         remove f from D;
15.         let removed := true;

16.         for each q⟨f_q, d1_q, d2_q⟩ ∈ S:
17.            if d1_q = f or d2_q = f then
18.               remove q from S;
19.               if (f_q = derived) then
20.                  add f_q to D;
21.               end if;
22.            end if;
23.         end for;
24.      end if;
25.   end for;
26. until (D = ∅) or (removed = false).
```

Table 2: Initial truth maintenance algorithm

for an explicit fact $d1_s = \emptyset$. In both cases, and when s was produced by a rule with a single clause in the premise, $d2_s = \emptyset$.

Furthermore, we introduce a set D wich contains *suspended* statements.

Definition 1. A *suspended* statement is a statement that is a candidate for removal from the repository during a transaction.

Initially, this set contains all statements f on which delete operations were performed in $\mathcal{T}$.

The initial truth maintenance algorithm is shown in table 2. The first loop (line 1-8) in the algorithm is an initialization step that labels each explicit statement that was removed in $\mathcal{T}$ as $derived$, and removes the corresponding justification from S. Notice that if a suspended statement is not explicit, it is removed from D, because derived statements can not be deleted except by deleting the explicit statements that justify them.

After this initialization, the algorithm enters a loop where it scans all suspended statements f (line 11). It determines the justification for each f (line 12), and if there is no justification then first the statement is removed from both B and D, and second each justification which contains the statement is removed as well (line 16-18). Since we are

removing justifications, the derived statements that were justified by these justifications need to be re-examined, so they are added to the set of suspended statements (line 19-21). The algorithm terminates when there are no more suspended statements, or when a complete pass is made without removing any statements from B.

4.3 Cyclic dependencies and grounded justifications

Unfortunately, the initial algorithm shown in table 2 fails to take the occurrence of cyclic dependencies in S into account. Consider, for example, the following set of RDF statements:

```
1. (rdfs:subClassOf, rdfs:domain, rdfs:Class) (derived, axiom)
2. (my:foo, rdfs:subClassOf, rdfs:Resource)   (explicit)
3. (my:foo, rdf:type, rdfs:Class)             (derived)
```

S now contains a.o. the following elements: $a\langle 1, \alpha, \emptyset\rangle$ (produced by RDF MT axiom assertion), $b\langle 2, \emptyset, \emptyset\rangle$ (produced by explicit assertion of statement 2), $c\langle 2, 3, \emptyset\rangle$ (produced by RDF MT rule 7a) and $d\langle 3, 1, 2\rangle$ (produced by RDF MT rule 2).

If now we execute an update transaction where statement 2 is removed, the TM algorithm first marks 2 as $derived$ and removes b from S. It then checks whether S contains a justification for statement 2. In the above, it finds it: c. The algorithm then concludes that statement 2 is still justified and does not remove it. This, however, is incorrect because c contains statement 3 which is itself dependent on statement 2 again (justification d). It is clear that 2 should have been removed and that justification c should not have been taken as sufficient evidence.

The problem lies in the fact that one of the justifying statements of c is itself dependent on the justified statement. Therefore, we introduce the concept of a *grounded* justification.

Definition 2. A justification $s\langle f_s, d1_s, d2_s\rangle \in S$ is called *grounded* if and only if neither $d1_s$ nor $d2_s$ are justified solely by f_s or by any other statement transitively justified solely by f_s.

The rationale behind this definition is simple: if a fact is explicit, it holds no matter what the other justifications for it are. If it is derived, however, the justification for it should not be completely dependent on itself.

We enhance the original algorithm to capture the problem of cyclic dependencies by, after labeling deleted statements as $derived$ and removing their corresponding justifications from S, computing a new set $G \subseteq S$ which contains the *grounded* justifications g. G is computed using a mark-and-sweep-like algorithm shown in table 3.

We begin with a set that contains only justifications for explicit facts (since these are always justified) and axiom facts (since these are never dependent on other statements) (line 1-5). Notice that we assume that the justifications produced by explicit assertion

```
01. for each s⟨f_s, d1_s, d2_s⟩ ∈ S:
02.   if (d1_s = α or d1_s = ∅) then
03.      add s to G;
04.   end if;
05. end for;
06. repeat
07.   let newfound := false;
08.   for each s⟨f_s, d1_s, d2_s⟩ ∈ S:
09.      if [(∃t⟨f_t, d1_t, d2_t⟩ ∈ G : f_t = d1_s)
                ∧ ((∃u⟨f_u, d1_u, d2_u⟩ ∈ G : f_u = d2_s) ∨ d2_s = ∅)] then
10.         add s to G;
11.         let newfound := true;
12.      end if;
13.   end for;
14. until newfound = false;
```

Table 3: Computing the grounded justifications

```
01. for each f ∈ D:
02.   if (f = explicit) then
03.      label f derived;
04.      remove s⟨f, ∅, ∅⟩ from S
05.   else
06.      remove f from D;
07.   endif;
08. end for;
09. determine G;
10. let E := S − G;
11. for each s_e⟨f_e, d1, d2⟩ ∈ E :
12.   add f_e to D;
13.   remove s_e from S;
14. endif;
15. for each f ∈ D:
16.   if (∀s⟨f_s, d1_s, d2_s⟩ ∈ G : f_s ≠ f) then
17.      remove f from B;
18.   end if;
19. end for.
```

Table 4: Mark-and-Sweep Truth Maintenance algorithm

for statements that are to be deleted are no longer present in S. From this basis, we iteratively add those justifications from S from which the justifying facts are already justified by G (line 6-14). This way, it is ensured that no justifications are added which are dependent on statements which themselves are no longer justified.

The enhanced TM algorithm is shown in table 4.

The algorithm in table 4 is a modification of the original algorithm: after labeling each deleted statement $derived$ and removing the corresponding justifications from S (line 2-8), the sets G (see table 3) and E are computed (line 9-10). Since we now have complete knowledge on justification for all statements in B, we can eliminate the

`repeat..until` loop from the original algorithm completely: all statements justified by expired dependencies are added to D (line 12) and all expired dependencies are removed from S (line 13). Then, all statements in D are verified, that is, they are deleted if and only if there is no justification for them in G (line 15-19). After looping over D once, the algorithm terminates.

4.4 Complexity

The algorithm presented in figure 4 has a complexity in the order of the number of deductive dependencies, $O(|S|)$. Since the size of S is determined by the number of entailments per statement, this number can become very high. This is a clear bottleneck and makes the algorithm less suited for the purpose of optimizing speed of delete operations: an alternative algorithm might avoid this bookkeeping task of computing and maintaining the deductive dependencies altogether.

However, since our other prime motivation for the algorithm is having the set of deductive dependencies available for higher-level reasoning tasks such as repository change tracking and statement-level security policies, there is a tradeoff to consider. Our current practical approach has been to accept the performance bottleneck in the computation of the set of dependencies in favor of enabling higher-level reasoning.

4.5 Implementation issues

In this section we will discuss several practical issues that arose when implementing the TM algorithm in the Sesame [3] [2] system.

4.5.1 Justification Inferencing

The algorithm tacitly assumes that the set of justifications, S, is known. However, like the closure of the set of statements, determining S requires inferencing over the set of RDF MT entailment rules from [5]. Unlike for the statement inferencing, however, optimizations that skip redundant inferences are not possible in this case, since it is important that S is complete.

In the test setup in Sesame, the determination of S is therefore implemented as a separate inferencing task. The justification inferencer is invoked after the complete closure has been computed and stored. It employs a basic backward chaining strategy: it loops over all statements and determines for each one whether it is a possible conclusion of an RDF MT entailment rule, and if so, which other statements satisfy the premise(s) for the matching rule. While this computation is expensive, it is only necessary to perform it once, at some time after new statements are added to the repository.

In the test setup, the justification inferencing is done directly after statements have been added, as part of the transaction. This means that the transaction takes longer. An

[3] See `http://sesame.aidministrator.nl/`

data set	explicit	closure	increase
OpenWine	4310	5289	23%
CIA	26285	30260	15%
SUO	4071	12498	206%
Wordnet	273681	373485	37%

Table 5: Increase in number of statements through closure computing

alternative approach would be to delay the justification inferencing, until the moment the TMS (or e.g. the versioning functionality) needs the information, or to do this part of the inferencing in a background process, locking delete operations until it is complete but leaving other types of operations on the repository available.

5 Results

We have run a number of tests, on computing the closure, and on general performance with both the brute-force approach and the justification-based approach, with the data sets introduced in section 3.2.

The tests were carried out using Sesame release 0.8, with the SQL92Sail and the TmsSQL92Sail for brute-force and justification-based inferencing respectively. In reporting the results we normalized the figures. This is done because our chief interest is comparing the two approaches rather than doing absolute performance tests, and also because the performance figures are highly dependent the version of Sesame, the used hardware, and the DBMS configuration.

5.1 Computing the closure

The overhead in storage space for the complete closure of the model for the data sets introduced in section 3.2 is shown in table 5. The algorithm for closure computing is an exhaustive forward chaining algorithm, using a few optimizations, as decribed in section 3.2.

In table 6 we see the comparative performance of closure computing, normalized against simple upload without closure computing.

We can observe from these result that adding our exhaustive forward chaining algorithm to the upload task adds an overhead, but that this overhead is not excessive.

5.2 Brute-force vs. TMS

In table 7 the comparative performance results for uploading data sets are shown, normalized against the brute-force approach. Uploading consists of inferring and storing

dataset	closure computing (norm)
CIA	1.45
SUO	1.41
Wordnet	1.13

Table 6: Performance overhead of closure computing, normalized against simple upload

data set	set size	add statement (norm)	total upload (norm)
OpenWine	5289	1.27	1.34
SUO	12498	10.50	10.25
CIA	30260	1.07	1.11
Wordnet	373485	1.36	1.36

Table 7: Performance of TMS-based statement adding, normalized against the brute-force approach

statements, and in the case of the TMS-enabled setup, it also includes computing the set of deductive dependencies.

As was to be expected, overall performance on upload is slightly worse in the justification-based approach, which is mainly caused by the computation of the set of deductive dependencies. Again, we notice that the SUO data set behaves differently from other sets due to its composition.

In table 8 the performance results for removing data are shown. The average given is the average per removal operation (note that this does not necessarily correspond to one removed statement, one removal operation may delete several statements).

From these results we can observe that justification-based removal performs significantly better than its brute-force counterpart, except on the SUO data set and the Wordnet data set, in which cases it performs worse.

data set	set size	remove (avg.) (norm)
OpenWine	5289	0.73
SUO	12498	1.08
CIA	30260	0.15
Wordnet	176037	1.68

Table 8: Performance of TMS-based statement removal, normalized against the brute-force approach

The reason the Wordnet and SUO sets perform worse when using the justification-based approach can be explained by the physics of the algorithm implementation and the testing environment. In the Sesame implementation of the TM algorithm, the set of grounded justifications G is computed using a rather complex SQL query that performs multiple joins with the table that stores the set of justifications, S. In the cases of SUO and Wordnet, this set of justifications is comparatively large – in the SUO test because there are so many new statements inferred (see table 5), in the Wordnet test because the data set itself is fairly large.

It turns out that the MySQL RDBMS used in the tests has limited capabilities for efficiently processing the SQL queries used to determine G when such large tables are involved, and its performance on such queries drops dramatically when the size of the table that stores D exceeds a certain threshold. The brute-force approach, which does not need to compute G, only has to use the SQL queries needed to do inferencing. However, these SQL queries are relatively simple and MySQL has less problems with evaluating these efficiently, even on large data sets. The upshot is that in these cases, re-computing the entire closure becomes more efficient than using the TMS approach.

Several strategies can be used to improve performance of the TMS, such as table optimization, indexing, or even the use of a more sophisticated query planner than the one available in MySQL. However, a more structured solution lies in the adjustment of the truth maintenance algorithm itself. In recent testing results, it has turned out that the strategy for computing the set G can be adapted to perform significantly better in situations where the number of removed statements in a transaction is lower than 10% of the total size of the data set. In future work we will further investigate and implement this strategy, and the implementation will be able to switch between strategies depending on the number of removed statements in a transaction.

6 Conclusions

In this article we have addressed several issues concerning RDF inferencing and truth maintenance. We have demonstrated that, contrary to earlier claims in the literature, exhaustive forward chaining is a feasible and even scalable approach for computing the deductive closure of an RDF(S) model. Also, we have argued that removal operations are important to consider in relation to the inference strategy employed, and we have presented an algorithm for capturing dependencies between statements and exploiting these dependencies for doing truth maintenance when statements are removed.

As we have shown, the exhaustive forward-chaining inferencing strategy performs very well on our selection of RDF datasets, both in speed and required storage space. In future work we will investigate how well this approach will scale to capturing the semantics of a language like OWL Lite [11]. The most obvious adjustment to the inferencing strategy to better deal with these more expressive languages, which we will investigate further in future work, is letting go of the strict forward chaining approach and building in some on-demand generation of entailed statements.

We have shown that the truth maintenance algorithm using dependencies between statements performs quite well on medium-sized data sets, and our goal of improving removal operation performance as compared to brute-force approaches is achieved on these data sets.

However, though the Sesame system itself can easily cope with data sets that consist of over 3 million statements, it seems apparent from the data obtained from – in particular – the Wordnet test that remove operations as implemented in the test system will not scale well to these high figures. A proviso here is that the test results were obtained using unoptimized code in the Sesame system, recent tests with more optimized code have actually significantly improved absolute performance, in many cases by a factor 10 or more. These results, however, are not presented here because they cannot be compared to the brute-force approach, for a very simple reason: the Sesame codebase now makes use of the TMS algorithm internally, and there is no brute-force counterpart implemented to obtain a valid comparison.

Regardless, we conclude that although this TMS approach performs satisfactorily for medium-sized (that is, up to approx. 200,000 statements) data sets and the current expressiveness of RDF and RDF Schema, the approach will have to be adjusted to cope with larger data sets. In particular, the justification-based truth maintenance algorithm can be further tuned to perform better under difficult circumstances by better indexing schemes and possibly using a more sophisticated query planner, as well as by adapting the computation of G, and switching strategies depending on the statistics of the transaction (in fact some of these optimization have already been implemented, resulting in the aforementioned performance increase).

Also, since not every task will require the higher-level services mentioned earlier, an alternative implementation of inference that is currently under investigation will make use of more sophisticated closure computing algorithms and will forgo keeping track of all deductive dependencies. This will result in the loss of capabilities for change tracking and security policies but will result in faster performance and a smaller memory footprint.

References

1. D. Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Candidate recommendation, World Wide Web Consortium, March 2000. See `http://www.w3.org/TR/2000/CR-rdf-schema-20000327`.
2. Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Ian Horrocks and James Hendler, editors, *Proceedings of the first International Semantic Web Conference (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science, pages 54–68, Sardinia, Italy, June 9 – 12, 2002. Springer Verlag, Heidelberg Germany. See also `http://sesame.aidministrator.nl/`.
3. John Davies, Dieter Fensel, and Frank van Harmelen, editors. *Towards the Semantic Web: Ontology-Driven Knowledge Management*, chapter 10: Ontology Middleware and Reasoning. Wiley & Sons, Europe, 2002. See also `http://www.ontotext.com/omm`.
4. Jon Doyle. A Truth Maintenance System. *Artificial Intelligence*, 12, 1979.

5. Patrick Hayes. RDF Model Theory. Working Draft, World Wide Web Consortium, January 23, 2003. See http://www.w3.org/TR/rdf-mt/20030123/.
6. Atanas Kiryakov, Kiril Iv. Simov, and Damyan Ognyanov. Ontology Middleware: Analysis and Design. On-To-Knowledge (IST-1999-10132) Deliverable 38, OntoText, March 2002. See http://www.ontotext.com/publications/index.html#KiryakovEtAl2002.
7. O. Lassila and R. R. Swick. Resource Description Framework (RDF): Model and Syntax Specification. Recommendation, World Wide Web Consortium, February 1999. See http://www.w3.org/TR/REC-rdf-syntax/.
8. Ora Lassila. Taking the RDF Model Theory Out for a Spin. In Ian Horrocks and James Hendler, editors, *Proceedings of the First International Semantic Web Conference, ISWC 2002, Sardinia, Italy*, number 2342 in Lecture Notes in Computer Science, pages 307–317. Springer-Verlag, Heidelberg, Germany, June 9 – 12, 2002.
9. J.P. Martins. The Truth, the Whole Truth and Nothing But the Truth. *AI Magazine: Special Issue*, 11(7), 1990.
10. J. A. La Poutré and J. van Leeuwen. Maintenance of transitive closure and transitive reduction of graphs. In *Workshop on Graph-Theoretic Concepts in Computer Science*, number 314 in Lecture Notes in Computer Science, pages 106–120. Springer-Verlag, Heidelberg, Germany, 1988.
11. Michael Smith, Chris Welty, and Deborah McGuinness. OWL Web Ontology Language Guide. Working draft, World Wide Web Consortium (W3C), March31 2003. See http://www.w3.org/TR/owl-guide.