Description Logic Reasoning for Dynamic ABoxes

Christian Halaschek-Wiener, Bijan Parsia, Evren Sirin,
and Adtiya Kalyanpur
Maryland Information and Network Dynamics Laboratory
8400 Baltimore Ave., College Park, MD, 20740 USA
{halasche,evren,aditya}@cs.umd.edu,bparsia@isr.umd.edu

1 Introduction

Recently, there has been interest in providing formal representation of Web content, which can then be processed using automated reasoning techniques. Due to data sources that produce fluctuating data, there exists a variety of description logic reasoning use cases which require frequent updates at the assertional level. These include prominent web services frameworks (e.g., OWL-S) that use description logics for service discovery and matchmaking, where devices register or deregister their descriptions (and supporting ontologies) quite rapidly. Additionally, Semantic Web portals often allow content authors to modify or extend the ontologies which organize their site structure or page content. Lastly, one of the common uses of description logic reasoners is in ontology editors. Most editors do not do continuous reasoning while one is editing, relying on an analogue of the *edit-compile-test* loop of most programming environments.

While there exists such use cases for reasoning under update, today's description logic reasoning services have been developed considering relatively static knowledge bases. In this paper, we investigate the process of incrementally updating tableau completion graphs created during consistency checking in the expressive description logics $\mathcal{SHOQ}(\mathcal{D})$ and $\mathcal{SHIQ}(\mathcal{D})$, which correspond to a large subset of the W3C standard Web Ontology Language, OWL-DL. We present an algorithm for updating completion graphs under both the addition and removal of ABox assertions; this provides a critical step towards reasoning over fluctuating/streaming data, as it has been shown that in KBs with substantially sized ABoxes, consistency checking can dominate reasoning time. We also provide an empirical analysis of the optimizations through an experimental implementation in an open source OWL-DL reasoner, Pellet.

2 Background

DL tableau-based algorithms decide the consistency of an ABox A w.r.t a TBox T (by TBox, we are additionally referring to all axioms for properties) by trying to construct (an abstraction of) a common model for A and T, called a *completion graph* [5]. Formally, a completion graph for an ABox A with respect to T is a directed graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{L}, \dot{\neq})$. Each node $x \in \mathcal{V}$ is labeled with a set of concepts $\mathcal{L}(x)$ and each edge $e = \langle x, y \rangle$ with a set $\mathcal{L}(e)$ of role names. The binary predicate $\dot{\neq}$ is used for recording inequalities between nodes. This graph is constructed by repeatedly applying a set of *expansion rules*.

For ABox updates, we consider addition and deletion of individual equality and inequality assertions $x = y$ and $x \neq y$, concept assertions $C(x)$ and role assertions $R(x, y)$. Updating ABox assertions in the presence of a TBox and an RBox brings up several issues with the semantics. For the purposes of this paper we use a straight-forward update semantics, which we call *Edit Semantics*. Intuitively, *Edit Semantics* can be described as an update in which all new ABox assertions are directly added (or removed) to the KB and no further changes/revisions are made. As the above example shows removing an assertion from the ABox under these semantics will not guarantee that the removed assertion will not be entailed anymore. Furthermore, an addition of a new axiom may cause inconsistencies in the KB as no additional changes can be made besides the addition or removal invoked by the update. Formally, we desribe this semantics as:

Definition 1 *(Edit Semantics) Let S be the set of assertions in an initial ABox A. Then under* Edit Semantics*, updating S with an an ABox addition (resp. deletion) α, written as $A + \alpha$ (resp. $A - \alpha$), results in an updated set of ABox axioms S' such that $S' = S \cup \{\alpha\}$ (resp. $S' = S \setminus \{\alpha\}$).*

This semantics might be a little unintuitive (especially from a belief revision point of view) but real world use of the update semantics is directly present in ontology editors and many document-oriented servers, e.g. Semantic Web Service repositories. For example, an OWL-S Web Service description can be seen as a set of ABox assertions and publishing/retracting this service description to/from a repository would be typically done under *Edit Semantics*. Edit semantics additionally aligns with the Formula based update semantics defined in [9].

3 Update Algorithm

The goal of the algorithm presented here is to update a previously constructed completion graph in such a way that it is still a model of the KB (if one exists).

The approach presented here is applicable to the Description Logics $\mathcal{SHOQ}$ [3] and $\mathcal{SHIQ}$ [4], as the difference between the two completion algorithms is independent of the update algorithm.

3.1 ABox Additions

Conceptually, tableau algorithms for $\mathcal{SHOQ}$ [3] and $\mathcal{SHIQ}$ [4] can be thought of as incremental in nature - *expansion rules* are applied in a non-deterministic manner to labels in the completion graph. Hence, new ABox assertions can be added even after previous completion rules have been fired for existing nodes and edges in the graph. After the addition, expansion rules can subsequently be fired for the newly added nodes, edges and their labels.

In order to update a completion graph upon the addition of a type assertion, $C(x)$, the algorithm first checks if the individual exists in the completion graph. If $x \notin \mathcal{V}$, then x is added to $\mathcal{V}$. Then C is added to $\mathcal{L}(x)$, if it does not already exist. If an individual inequality relation, $x \neq y$, is added to the completion graph, the algorithm checks if $x \in \mathcal{V}$ and $y \in \mathcal{V}$. If either do not exists, then they are added to the graph. Additionally, if the $x \neq y \notin x \dot{\neq} y$, then it is added. Alternatively, if an individual equality relation, $x = y$, is added to the completion graph, the algorithm checks if $x \in \mathcal{V}$ and $y \in \mathcal{V}$. If either do not exists, then they are added to the graph. Additionally, x and y are merged. Lastly, if a role, $R(x, y)$, is added to the ABox and $\langle x, y\rangle \notin \mathcal{E}$, then $\langle x, y\rangle$ is added to $\mathcal{E}$ and R is added to $\mathcal{L}(\langle x, y\rangle)$. If however, $\langle x, y\rangle \in \mathcal{E}$ but $R \notin \mathcal{L}(e)$, then R is added to $\mathcal{L}(x)$.

After the graph has been updated, the completion rules are reapplied to the updated completion graph, as the update may induce additional expansion rules to be fired. We note here that if there has previously been a deletion update, then previously explored branches which had a clash must be re-explored as the deletion could have removed the axiom which caused the clash.

3.2 ABox Deletions

When updating a completion graph under ABox axiom removals, *components* (nodes, edges, labels, etc.) in the graph that correspond to the removed axiom cannot be simply removed. This is because as completion rules are applied, newly added portions of the graph are dependent on the presence of the original axioms in the KB. By deleting an ABox assertion, components of the graph that are dependent on that assertion need to be updated as well.

In order to account for this, we propose using axiom pinpointing [6], which tracks the dependencies of graph components on original source axioms from the ontology through the tableau expansion process. More specifically, the application of the expansion rules triggers a set of *events* that change the state of the completion graph, or the flow of the algorithm. In [6] a set of change events is defined, including adding labels to nodes and edges, etc. In order to record the changes on the completion graph, [6] introduces a *tracing function*, which keeps track of the asserted axioms responsible for changes to occur. The *tracing function*, τ, maps each event $e \in \mathcal{E}$ to a collection of sets, each set containing a fragment of K (axioms) that cause the event to occur. This tracing function is maintained throughout the application of tableau expansion rules defined in [5].

For purpose of this work, the original set of *change events* has been extended [2] to include all possible events that can occur during the application of expansion rules. The extension of events includes removing nodes, edges, labels (in case of merges), adding nodes and edge, etc. Additionally the tracing function maintenance through expansion rule application has been extended. Although the tracing extension has been provided for $\mathcal{SHOIQ}$, it is trivial to see how they are applied to $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$ completion strategies. For brevity, further details are omitted, however they can be found in [2].

In general, the update algorithm for deletion works as follows. When an ABox axiom is removed, the algorithm performs a lookup in the graph for all *change events* whose axiom traces include the axiom number of the deleted axiom. These events are *rolled-back* if and only if their axiom trace sets only include sets which contain the deleted axiom, possibly among other axioms. By *roll-back* we refer to simply undoing (the inverse) the event (e.g., rolling back the event $Add(x, \mathcal{V})$ would be the process of removing x from $\mathcal{V}$). If there exists additional axiom traces for that particular event that do not include the removed axiom, then only the sets including the removed axiom are deleted from the axiom trace set; in this situation the actual event is not *rolled-back*. This is intuitive as there still exists support for that particular event.

As in the approach for additions, the completion rules must be reapplied to the updated completion graph as it is possible for the graph to be incomplete for the KB. Axiom tracing additionally requires a slight modification to the update approach for ABox additions. For example, in the case that a individual type assertion is added to the KB, the algorithm must add a new tracing set to the axiom trace for the affected components (this set will consist of the new axiom number). The update algorithm ($UPDATE(G, \alpha)$) is provided in Figure 1. Note that *deps* (dependents) is the inverted tracing function index. Additionally, the operator $\uplus$ is defined as follows: let $\mathbf{S_1} = \{S_1^1, ..., S_m^1\}$ and $\mathbf{S_2} = \{S_1^2, ..., S_n^2\}$ be two collections of sets, then:

$$S_i^1 \ (1 \leq i \leq m) \in S_1 \uplus S_2 \text{ iff } S_i^1 \not\subset S_j^2 \text{ for all } j,\ 1 \leq j \leq n$$
$$S_j^2 \ (1 \leq j \leq n) \in S_1 \uplus S_2 \text{ iff } S_j^2 \not\subset S_i^1 \text{ for all } i,\ 1 \leq i \leq m$$

Due to space limitation soundness proofs for the update algorithm are omitted, however they can be found in a more detailed technical report[1].

[1]Description Logic Reasoning for Dynamic ABoxes, UMIACS Technical Report. Available

```
function UPDATE( G, α )
  if α is an addition then
    if α is a x ≠ y or x = y then
      let op be the operation, such that op ∈ {=, ≠}
      if x ∉ 𝒱 then
        𝒱 ← 𝒱 ∪ {x}
      if y ∉ 𝒱 then
        𝒱 ← 𝒱 ∪ {y}
      if op is ≠ then
        if x ≠ y ∈ x≠̇y then add it
      if op is = then
        Merge x and y
      τ(x op y) ← τ(x op y) ⊎ {{α}}
      τ(Add(x, 𝒱)) ← τ(Add(x, 𝒱)) ⊎ {{α}}
      τ(Add(y, 𝒱)) ← τ(Add(y, 𝒱)) ⊎ {{α}}
      deps(α) ← deps(α) ∪ {{x op y}{Add(x, 𝒱)}{Add(x, 𝒱)}}
    else if α is a individual type addition, (C(x)) then
      if x ∉ 𝒱 then
        𝒱 ← 𝒱 ∪ {x}
      ℒ(x) ← ℒ(x) ∪ {C}
      τ(Add(x, 𝒱)) ← τ(Add(x, 𝒱)) ⊎ {{α)}}
      τ(Add(C, ℒ(x))) ← τ(Add(C, ℒ(x))) ⊎ {{α}}
      deps(α) ← deps(α) ∪ {{Add(x, 𝒱)}{Add(C, ℒ(x))}}
    else if α is a role assertion addition, R(x,y) then
      if ⟨x, y⟩ ∉ ℰ then
        ℰ ← ℰ ∪ {⟨x, y⟩}
      ℒ(⟨x, y⟩) ← ℒ(e) ∪ {R}
      τ(Add(x, 𝒱)) ← τ(Add(x, 𝒱)) ⊎ {{α)}}
      τ(Add(y, 𝒱)) ← τ(Add(y, 𝒱)) ⊎ {{α)}}
      τ(Add(⟨x, y⟩, ℰ)) ← τ(Add(⟨x, y⟩, ℰ)) ⊎ {{α)}}
      τ(Add(R, ℒ(⟨x, y⟩))) ← τ(Add(R, ℒ(⟨x, y⟩))) ⊎ {{α}}
      deps(α) ← deps(α) ∪ {{Add(x, 𝒱)}{Add(y, 𝒱)}{Add(⟨x, y⟩, ℰ)}{Add(R, ℒ(⟨x, y⟩))}}
    Apply expansion rules to G
    if there is a clash then
      Perform backjumping
  else if α is a deletion then
    events ← deps(α)
    deps(α) ← ∅
    for each e ∈ events do
      traces ← τ(e)
      for each t ∈ traces do
        if α ∈ t do
          traces ← traces \ t
      if traces = ∅ do
        roll-back e
      τ(e) ← traces
    Apply expansion rules to G
    if there is a clash then
      Perform backjumping
  return G
```

Figure 1: Pseudo-code of update procedure for $\mathcal{SHOQ}(\mathcal{D})$ and $\mathcal{SHIQ}(\mathcal{D})$ ABoxes

4 Implementation and Evaluation

We have implemented the approach presented in this paper in an open source OWL-DL reasoner, Pellet [2]. In order to evaluate our update algorithm, we have performed an emperical evaluation using two different KBs with large ABoxes - the Lehigh University Benchmark (LUBM)[3] and AKT Reference Ontology [4].

Three tests were run over three KBs consisting of one, two, and lastly three universities from the LUBM dataset generator. First an initial consistency check was performed and then in each test, a random update was selected which was used to update the KB. In the regular version of the reasoner, the cached completion graph was discarded, while in the optimized reasoner the update algorithm was utilized. For each KB size, varying sized additions and deletions were randomly selected from the dataset. Update sizes include single axiom, twenty-five axioms, and fifty axioms (individual and/or role). Each test was performed twenty-five times and the results were averaged. Expressivity and KB statistics are provided in Table 1. Results for additions and removals in the LUBM

Name	Classes / Properties / Individuals / Assertions	Expressivity
LUBM-1 Univ	43 / 32 / 18,257 / 97,281	$\mathcal{SHI}$
LUBM-2 Univ	43 / 32 / 47,896 / 254,860	$\mathcal{SHI}$
LUBM-3 Univ	43 / 32 / 55,110 / 295,728	$\mathcal{SHI}$
AKT-1	160 / 152 / 16,505 / 70,948	$\mathcal{SHIF}$
AKT-2	160 / 152 / 32,926 / 143,334	$\mathcal{SHIF}$

Table 1: LUBM and AKT Portal Dataset Statistics

test are presented in Figure 2 (timing results are shown in milliseconds and the scale is logarithmic). We note that the '0' axiom value represents the initial consistency check. In both versions of the reasoner, initial consistency checks are comparable. However for both update types, performance improvements ranging from one to three orders of magnitude are achieved under updates in the reasoner with the optimized update algorithm. This is due to the avoidance of re-firing of completion rules by maintaining the previous completion graph; therefore very few (if any in some cases) completion rules must be refired. This provides direct empirical evidence for the effectiveness of the update algorithm. In a second evaluation, two datasets[5] adhering to the AKT Reference ontology were used (statistics shown in Table 1). The tests were structured in the same manner as the LUBM test. Again, each test was performed twenty-five times and the results are averaged over these iterations. All timings are in milliseconds and the scale is logarithmic. Similar to the LUBM test, update performance is

at http://www.mindswap.org/papers/2006/aboxUpdateTR2006.pdf

[2]Pellet OWL-DL Reasoner: http://www.mindswap.org/2003/pellet/

[3]LUBM Ontology: http://swat.cse.lehigh.edu/projects/lubm/

[4]AKT Ontology: http://www.aktors.org/publications/ontology/

[5]Hyphen-REA: http://www.hyphen.info/rdf/hero_complete.zip

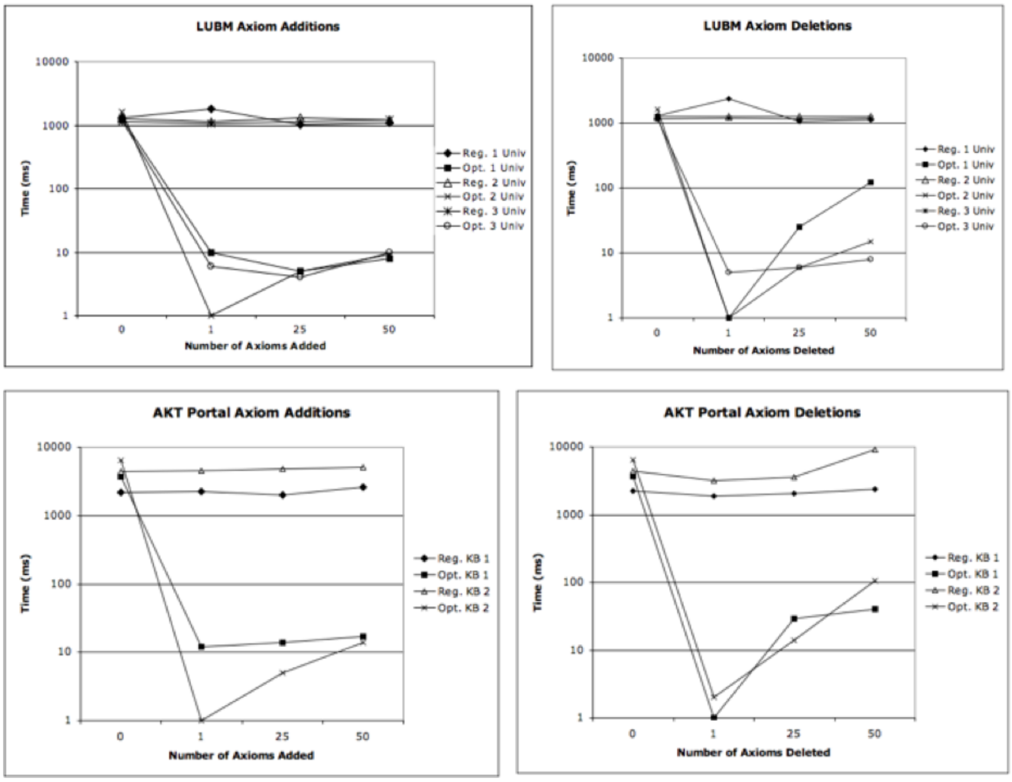

Figure 2: Addition and Removal Updates of LUBM and AKT datasets

improved between one to three orders of magnitude. Note that the performance of the update is better in the LUBM test cases; this is primarily due to the increased complexity of the AKT Reference Ontology. Therefore, a larger number of expansion rules are applied after the update. However the update algorithm greatly outperforms the regular reasoner, again demonstrating the effectiveness and overall impact of the update algorithm.

5 Related Work and Conclusion

To our knowledge there has been no previous work in DL reasoning algorithms for incremental maintenance of completion graphs. We do note that this work can be paralleled to view maintenance and truth maintenance; however we deal with a more expressive logic and a different proof theory. There has also been recent work in optimizing incremental instance retrieval in expressive description logics [1]; however previous work has not addressed the notion of updating the completion graphs for consistency checking. Additionally, in [7, 8] the authors specify update semantics for descriptions logics; however this work is less related to belief revision and is independent as we are concerned with maintaining the internal state of the reasoner.

In this paper, we have presented an algorithm for updating completion graphs for the Description Logics $\mathcal{SHIQ}(\mathcal{D})$ and $\mathcal{SHOQ}(\mathcal{D})$ under both the addition and removal of ABox assertions, providing a critical step towards reasoning procedures for fluctuating or streaming data. We have provided an empirical analysis of the algorithm through an experimental implementation in the Pellet reasoner. Our initial results are very promising as they demonstrate orders of magnitude performance improvement.

References

[1] Volker Haarslev and Ralf Moller. Incremental query answering for implementing document retrieval services. In *International Workshop on Description Logics*, pages 85–94, 2003.

[2] Christian Halaschek-Wiener, Aditya Kalyanpur, and Bijan Parsia. Extending tableau tracing for abox updates. In *UMIACS Tech Report*, 2006. http://www.mindswap.org/papers/2006/aboxTracingTR2006.pdf.

[3] I. Horrocks and U. Sattler. Ontology reasoning in the SHOQ(D) description logic. In B. Nebel, editor, *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, 2001.

[4] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of the 6th Int. Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.

[5] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for SHOIQ. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*. Morgan Kaufman, 2005.

[6] Aditya Kalyanpur, Bijan Parsia, Bernardo Cuenca-Grau, and Evren Sirin. Tableau tracing in shoin. UMIACS Tech Report.

[7] H. Liu, C. Lutz, M. Milicic, and F. Wolter. Updating description logic aboxes. In *International Conference of Principles of Knowledge Representation and Reasoning(KR)*, 2006.

[8] Mathieu Roger, Ana Simonet, and Michel Simonet. Toward updates in description logics. In *International Workshop on Knowledge Representation meets Databases*, 2002.

[9] M. Winslett. Updating logical databases. In *Updating Logical Databases*. Cambridge University Press, 1990.