EVOWIPE SparqlUpdater Documentation

Contents

1 Overview

This document describes a tool providing functionality for automatically performing SPARQL-DL updates on OWL ontologies. It does not describe basics of the Semantic Web or ontology processing (which includes core terms like A-Boxes), and neither does it explain details of the SPARQL-query language. Such knowledge should naturally be given before reading this documentation, although only rough basics should be necessary.
The Pellet OWL-DL Reasoner[3] is used for computation of justifications (i.e. explanations) for axioms in the ontology, as well as for entailment checking. Apart from that, the tool is written in Java 8 and uses the OWL API 4.0[1] for operations on ontologies.
There are a variety of ways to perform deletions in order to ensure that an optimal amount of insertions and deletions are ultimately successful. We employ so-called *update semantics* to define the preferred way of performing a contraction. Usually, a deletion will be performed by identifying minimal hitting sets (see section 2.1) of axioms, and one hitting set is chosen for performing the deletion. For each possible hitting set, the insertion will be done on the resulting ABox after performing the deletion. If any inconsistencies within the knowledge base arise as a consequence, a debugging step must be performed in order to remove the inconsistency. The resulting pair of hitting sets, one for the deletion and one for the debugging step, is generally referred to as an *implementation.*
Usage of the tool will be described briefly in section 3, while the next section 2 gives an overview of the possible semantics for performing an update. Lastly, a short overview is given on the environment installation process in section 4.

2 Semantics

The heart of the Sparql-Updater is the implementation of various semantics used for choosing a way (i.e. an implementation) to perform an update. Each way is described briefly in the following sections.

2.1 Prerequisite: Hitting Sets

In order to perform a deletion, it is mandatory to find *minimal hitting sets* of axioms to delete in order for the axiom not to be entailed by the ontology after deleting all chosen axioms. Each hitting set represents one way to perform the deletion in such a manner. However, which of the hitting sets provided by the reasoner should be used must be decided by using one of the defined semantics. Some semantics will even add more axioms to be deleted, depending on their use cases.

2.2 Maxichoice and Meet Semantics

On the one hand, it is possible to choose a minimal deletion w.r.t. set inclusion. This *maxichoice* semantics therefore disregards all hitting sets that are supersets of others. Thus, the resulting ABox is maximal w.r.t. set inclusion.
On the other hand, an approach can be considered where the resulting ABox has minimal cardinality. This is called *meet* semantics. It is rather pessimistic and will practically merge all minimal hitting sets.
If an insertion is performed, either of the above semantics can be chosen to perform the debugging step. That said, it is entirely possible to use any combination of the above semantics to perform deletions and debugging steps.

2.3 Query-driven Semantics

These semantics aim to maximize the sum of successful deletions and insertions. As such, deletion and debugging hitting sets are closely tied together. The way this is done is for each possible hitting set, an optional (if there is an inconsistency) debugging step is performed hypothetically, and if the amount of successful deletions and insertions - i.e. the amount of to-be-deleted axioms not entailed combined with the amount of to-be-inserted axioms that are entailed by the ontology - is the current largest, this implementation is chosen.
This approach is entirely based on cardinality. A much more desirable way to perform a query-driven update is to combine successful deletions and insertions for each implementation in a set of axioms and to choose the implementation that is maximal w.r.t. set inclusion. In other words, any implementation is discarded that has a combined set of successful deletions and insertions that is a subset of another implementation's set of the same nature.

2.4 Rigid Semantics

Rigid semantics tries to minimize deletion of rigid concept assertions. A concept assertion is one of the type

```
<individual> type <class>
```

or (different notation)

```
<individual> a <class>.
```

Rigid concepts are denoted in OWL-RDF as such:

```
<rdfs:comment>rigid</rdfs:comment>
```

These comment annotations need to be added manually. A rigid type assertion is a very fundamental statement about an individual. Examples would be teachers being persons. As a consequence, deleting such assertions - if done through a hitting set without the actual original intend to delete the class assertion - may unintentionally upset the balance of the ontology. Rigid semantics attempt to minimize such deletions to maximize the amount of rigid class assertions in the updated knowledge base.

In fact, in many cases it is desirable to completely remove the individual from the knowledge base as a consequence of deleting its rigid class assertion. Imagine a teacher no longer being a person. This usually makes little practical sense or use, which is why rigid semantics will optionally delete all ABox axioms containing the respective individual. Afterwards, a debugging step is performed as usual for each hitting set. In the resulting union of deleted axioms (including deletions for the debugging step), a check is made to find the implementation that minimizes rigid deletions. This is done through subset relations as usual (depicted in section 2.3).
Please note: If additional deletions are performed, recursively checking whether the debugging step (performed afterwards) introduces any more rigid deletions would be required (in theory). This is currently not done.

2.5 Dependency-Guided Semantics

A concept is said to be dependent on another concept w.r.t. some data or object property if it has a comment annotation of this kind:

```
<rdfs:comment>dependOn C w.r.t. q</rdfs:comment>
```

Again, this has to be added manually to the ontology.
If a class assertion with a dependency is deleted from the ontology, a cascade operator may be defined in such a way (note: not a precise definition) that it will delete other class assertions related to classes dependent on the respective class.
Trying to minimize the amount of cascaded deletions due to dependencies results in the Dependency-Guided Semantics. Again, choosing an implementation is done through subset relations.

2.6 Combination of Rigidity-Guided and Dependency-Guided Semantics

This combination of rigidity- and dependency-guided semantics addresses the problem of deleting rigid assertions when performing a cascading step for dependency-guided contraction. This in turn could lead to further violated dependencies, leading to potentially recursive behaviour until no more dependencies are violated and all rigid deletions have properly been taken care of (as in rigidity-guided semantics, see 2.4). With these semantics, all possible ways to perform such a contraction will be determined. In the end, a minimal implementation w.r.t. set inclusion is chosen for the final deletion.

3 Usage

3.1 General

The Sparql-Updater-Tool must be called through java:

```
foo@bar:<path-to-sparqlupdater-target-folder>$ java -jar
    sparqlupdater-0.0.1.jar
```

The following command line options are available:

```
 -b,--benchmark           Perform benchmarks
 -c,--cascade <arg>       Use cascading behaviour for
    update semantics
 -c1,--custom1 <arg>      First option for custom update
    semantics (for use
                          with -s custom ONLY)
 -c2,--custom2 <arg>      Second option for custom update
    semantics (for use
                          with -s custom ONLY)
 -f,--full                Benchmark all possible axiom
    combinations
 -h,--help                Show help
 -i,--input <arg>         Ontology input file path
 -m,--method <arg>        Method that will be used to
    generate explanations
                          (Default: glass)
 -o,--output <arg>        Output file path for ontology/
    benchmark stats
 -q,--query <arg>         Query/update input file path
 -s,--semantics <arg>     Semantics for choosing deletions/
    implementations.
                          Options: custom, query_set,
                              query_car, rigid,
                          depend, rigdep
 -t,--timeout <arg>       Timeout in seconds (default: none
    )
 -u,--user-input          Ask for user input
 -v,--verbose <arg>       Level of talkativeness (0-4,
    default: 1)
 -x,--max <arg>           Maximum number of generated
    explanations for each
                          inference (Default: 100)
```

Please note that the *-c* option is not properly implemented. Cascading behaviour works only with dependency-guided and rigidity-dependency-guided-semantics in the current state.
The term *custom semantics* refers to a combination of the *meet* and *maxichoice* semantics as described in section 2.2. The terms *query_set* and *query_car* refer to query-driven semantics maximizing the amount of successful deletions and insertions based on set inclusion and cardinality respectively.
Usually, a verbosity of 0 will only show results and user input prompts, while verbosity 1 should be the general option (and default) to use for users. Any

higher verbosity is mostly for debugging purposes or meant for advanced users in general having a basic understanding of how the semantics are implemented. The application will only ask users to choose an implementation if the *-u* argument is given. In this case, users will also be asked whether or not to perform deletions of individuals for rigid semantics as described in section 2.4.
The *-b* and *-c* options do not actually require another argument and can be provided without further ado. For benchmark mode, the *-f* option will try to delete any axiom possible within the ontology, which takes a considerable amount of time. Otherwise, only entailed axioms are attempted to be deleted. You may use *-m black* as an alternative to the default glass method.
If you don't provide an argument for the output option, a default name is generated. **Please note**, however, that existing files with the same name are never overwritten.
If not specified otherwise, options require integers as additional arguments.
Example call:

```
foo@bar:<path-to-sparqlupdater-target-folder>$ java -jar
    sparqlupdater-0.0.1.jar -i tests/test_simple.owl -q
    tests/test_simple.sparql -o test_simple_out.owl -s
    custom -c1 max -c2 meet -x 10 -u
```

3.2 Benchmarking

We have included a benchmarking mode that will not use a given input query for deletions. Instead, it will iterate over all individuals, properties and concepts in the ontology and try to perform a deletion for each combination thereof (with given semantics). That said, there are no insertions during this process. The results are stored to the provided output file path, or a default name will be generated for the output file that ends with "_stats.txt." and that contains the ontology and semantics used.
In the output file, one line can be found for each deletion that was performed. The general template is:

```
Axiom    Semantics    Amount of hitting sets    Number of
     axioms in each hitting set
```

An example:

```
tom a StudentTrainee    depend    3|4|3|4
```

Here, the axiom *"tom a StudentTrainee"* was deleted from the input ontology with dependency-guided semantics. This resulted in 3 possible ways (i.e. hitting sets of axioms) to perform the deletion, the first and fourth containing four axioms each, and the second hitting set consisting of three axioms to be deleted.
Please note that for rigid semantics, optional deletions are automatically performed in benchmark mode. By default, only entailed axioms are attempted to be deleted. If this is not desired, the *-f* argument should be supplied.

4 Installation

4.1 Requirements/Dependencies

1. Operating system

   So far, the system has been used on Ubuntu 14.04.5 LTS exclusively. There has been no experience deploying it on different operating systems. Other Linux distributions should be fine. Windows should technically work just fine as well as long as there is a way to provide command line options.

2. Java

   Naturally, Java 8 or higher is required to run or build the program.

3. Pellet

   The Pellet reasoner[3] is absolutely required if you are trying to build the program from source code. Otherwise, it should not be a requirement.

   `https://github.com/stardog-union/pellet`.

4. Maven

   For anything else, there is an Apache Maven[2] *pom.xml* configuration file provided in the repository that should automatically download any other required libraries. Keep in mind however that Pellet will have to be added externally to your local Maven configuration, and thus the *pom.xml* will likely need to be adjusted accordingly to reflect your installation.

4.2 Instructions

For simply running the distributed program, which should - in its basic form - just be a .java file, you do not actually require anything else! You are all set up and can start using it the way depicted in section 3.
However, keep in mind that not all ontology formats are supported; common OWL RDF should work fine for starters.
If building from source, it is required to build Pellet first and foremost. In my case, the cloned Pellet directory then had a folder named "dist/lib" which included .jar files for each Pellet module that I could import in Maven. The *pom.xml* reflects the local configuration on our system. This approach is somewhat clumsy. Pellet should also come with its own *pom.xml* files that could naturally be imported as well to build everything automatically along with our Sparql-Updater. There really are a variety of ways to do this. The important step remains importing the individual Pellet modules into Maven so that they can be used during the compilation process.

5 Future Work

Finally, there are many ways the existing work could be improved on; firstly, the output provided by the updater could be made much more fancy and self-

explanatory. This documentation could use some more detailed explanation on individual concepts of the Semantic Web, semantics for choosing implementations, and implementation details. There also are more semantics that could be implemented, such as provenance-based semantics. A few shortcomings of existing semantics have been mentioned, such as rigid semantics not performing recursive checks for rigid deletions, so naturally there is room for improvement in the code. Naturally, if any bugs are ever encountered, those should be fixed as soon as possible.
For any remarks, unintelligible error messages (perhaps with new insights?), problems or frustration during installation and other stuff, shoot me a mail: dvolz@uni-koblenz.de

References

[1] The OWL API. `http://owlcs.github.io/owlapi/`.

[2] The Apache Software Foundation. Welcome to Apache Maven. `https://maven.apache.org/`.

[3] Complexible Inc. Pellet: An Open Source OWL DL reasoner for Java. `https://github.com/stardog-union/pellet`.